Make Learning Fun with Programming Contests

Gines Garcia-Mateos and Jose Luis Fernandez-Aleman

Department of Informatics and Systems,
University of Murcia, 30100 Espinardo, Murcia, Spain
{ginesgm,aleman}@um.es

Abstract. Usually, higher education teachers have to deal with highly populated classes and low levels of motivation. Making more entertaining courses is a good way to overcome these limitations. But, how can fun and entertainment be introduced in a course which is mainly based on a final exam evaluation? We propose a new methodology based on two key ideas: (i) replacing the final exam with a series of activities in a continuous evaluation context; and (ii) making those activities more appealing to the students. We describe an e-learning experience carried out in a second-year programming course for computing majors. The activities are designed as on-line programming competitions, where all students participate and are able to see their global ranking. Experimental results show the effectiveness of this approach. On average, the dropout rate decreased from 72% to 45% while the pass rate doubled.

Keywords: Learning strategies, programming contests, e-learning, on-line judging.

1 Introduction

Since 1995, the percentage of students who have completed secondary education has increased in OECD countries, on average, a 7%. The number of graduates in higher education has also grown [1]. However, while demand for human resources in science and technology has increased in these countries (it represents between 25% and 35% of the total labour force), the number of university graduates in engineering and science –excluding health and welfare– has declined to one-fifth [2]. Although in absolute terms the number of enrollments has risen, science and engineering degrees are less and less attractive and dropout rates continue to be high. To boost the supply of scientists and engineers, OECD offers some recommendations. One of these suggestions is to make science and engineering more accessible and attractive to young students.

Edutainment is an emerging alternative to traditional education methods. Rapeepisarn *et al* [3] pointed out that "edutainment is the act of learning heavily through any of various media such as television programs, video games, films, music, multimedia, websites and computer software". Entertainment is the media to help students learn. The approach we describe here is in line with this goal: raising interest of the students by *virtually* transforming a computer science course into an on-line programming contest.

Z. Pan et al. (Eds.): Transactions on Edutainment II, LNCS 5660, pp. 246–257, 2009.

In our proposal, a traditional "final exam" methodology is replaced by a series of carefully designed activities, many of them organized as programming assignments. The key element of this e-learning course on programming is a web-based automatic judging system called Mooshak [4], which was originally created to manage on-line programming contests. Preliminary results show the viability of the learning experience, and a high capacity to generate motivation and enthusiasm among students. The approach is highly complementary with other learning techniques and could be applied to other courses, specially in a computer science degree.

The rest of the paper is organized as follows. Section 2 presents a brief review of some related work. Then, we introduce the methodological principles underlying our proposal in section 3. Section 4 describes the main results of the edutainment-based methodology applied to 337 students in a second-year course for computing majors. Section 5 discusses the results obtained by employing this new e-learning method. Finally, in section 6, we present some concluding remarks and outline the efforts to be made in the future.

2 Related Work

Edutainment has been successfully used in robotics, mathematics, language learning and many other areas. Our paper focuses on computer science, particularly programming. Computer programming is in the core knowledge of many science and engineering degrees. In the literature, most authors reach the same conclusion: learning to program is difficult [5].

Therefore, many techniques and methods have been proposed to improve novice students comprehension in teaching programming [5]. E-learning constitutes a viable and promising alternative in programming pedagogy.

A good example is Guerreiro and Georgouli [6,7], who proposed an e-learning educational strategy in first-year programming courses. They adopt Mooshak automatic judging system for grading lab assignments and for self-assessment purposes; some sample views of Mooshak are shown in Figure 1. This automatic evaluation accounts for about 30% of the final mark. The approach provides important benefits in CS1. A well thought out set of test cases prevents wrong programs sent by students from passing test runs. As a consequence, students must be much more rigorous in developing their programs. Likewise, students obtain immediate feedback from Mooshak. Another advantage of the proposal is the objectivity of the evaluation. Moreover, the authors consider that teachers can save time and work if an automatic judging system is used. Nevertheless, important concepts such as robustness and legibility are manually graded by the instructors.

Our novel contribution resides in *how* to apply the on-line judging system: we take Guerreiro and Georgouli's strategy one step further, by completely replacing the traditional "final exam evaluation" with a series of activities, most of them using Mooshak. Thus, two important benefits are obtained: the students are very motivated to take part in the proposed activities, with the hope of avoiding the

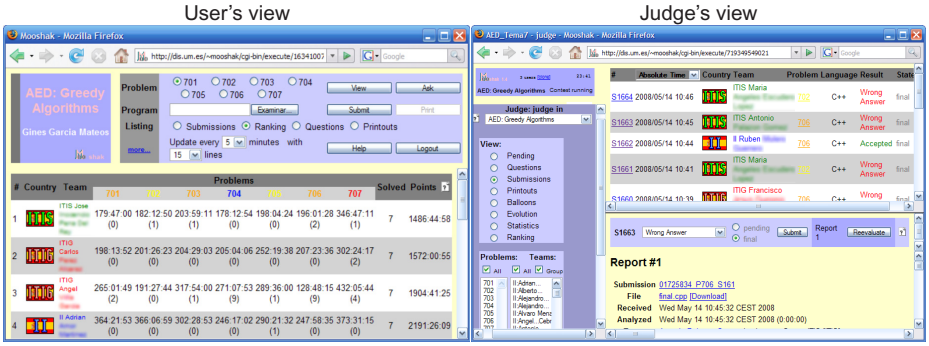

Fig. 1. Two sample views of Mooshak [4]. Left: sample view for a user of the system (the students). The user can access the description of the problems, the list of submissions done by all the users, the ranking of the best students, and the questions done and answered. Right: sample view for a judge (a teacher). The judge is able to see and analyze the submissions done, rejudge submissions, answer questions, and view statistics of system's usage.

final exam; and the work of the students is evaluated along the course, rather than just in a single final exam. To ensure the authorship of the programs, a source code plagiarism detection environment [8] is used. To appraise the quality of the code, the human factor remains to be prominent.

According to some authors, edutainment can be organized in different ways [9,10,11]. Our experience can be classified as follows:

Formal learning: It takes place in an education institution, leading to recognized qualifications and uses organized learning situations.

Interactive and participatory: Students play and participate in the *game*.

Motivation: All students share the same interest: learn to program.

Type of media: The game type is a competition where edutainment is accomplished by a web-based educational system.

3 Methodological Approach

The pioneering experience described here was applied to a second-year course on programming for computing majors; its duration is annual (two consecutive semesters). The main problem observed in this course was a low motivation and participation of the students, that resulted in very high dropout rates. Approximately between 2/3 and 3/4 of the enrolled students dropped out.

With the aim of reversing this trend, we developed a more participatory methodology, based on a continuous evaluation organization, with activities that are appealing and motivating for all students. In this section we describe in detail the key elements of the proposed approach. First, we establish the pedagogical principles that have guided our efforts. Then, the basic aspects of on-line judging systems are presented. Finally, we analyze how to introduce this tool into the learning process.

3.1 Pedagogical Principles

By analyzing our teaching experience in previous years, and also taking into account some recommendations given by other authors [5], we have identified the following pedagogical principles:

Motivation. With a dropout rate around 70%, increasing the motivation of students is essential. By *motivation* we understand the desire to learn new concepts and methods, and to put them into practice.

Active learning. The students have to be involved in, and conscious of, their own learning process. An active methodology, where students are not mere spectators, is necessary to achieve a real and long-lasting learning.

Autonomous work. We believe that the best way to learn computer programming is by programming. A simple memorization of concepts is nearly useless. The students have to reflect on algorithmic problems by themselves and create autonomously their programs.

Feedback of the learning process. This feedback is considered from the point of view of the students. Traditionally, they just obtain a final mark at the end of the course. The method should provide students with a continuous evaluation on how well they are doing.

All of these objectives require more entertaining and participatory activities, both in class and out of class. In these situations is where e-learning tools can produce a great benefit, not by substituting the teachers, but by helping them to control and evaluate the activities.

3.2 On-Line Judging

The key element of our approach is an on-line judging system. This system is an automatic tool which is able to evaluate the correction of computer programs, based on a predefined set of pairs input/output. It has a web-based interface –shown in Figure 1–, which is different for the students, teachers, guest users and the system administrator.

More specifically, we are currently using Mooshak 1.4 [4], which is free and publicly available. This system was originally created by Jose P. Leal to manage programming competitions. However, Mooshak is applied more and more to computer programming learning. It works as follows:

- A set of problem descriptions is available in the students' web interface for each activity that is proposed. These descriptions present problems related with the theoretical concepts studied in class. Each description contains a statement of the problem, a precise specification of the input of the program and the expected output, along with some sample input/output pairs.
- The students can select any problem from the set to solve it. First, they tackle the problem in their own computers, by writing a program which efficiently produces the expected outputs. When they have tested their implementation enough, they submit the solution to the judge using their interface.

- The on-line judge receives the source code, compiles the program, and executes it using the predefined sets of secret input cases. Then, Mooshak analyzes the output of the program (comparing it to the expected output) and sends a response to the student indicating whether the program is correct or not. If the program is not accepted, the judge reports about the rejection cause, such as for example if the program has a syntax error, or it is inefficient in time or memory.
- Statistical information of all the submissions sent to the judge is accessible both for teachers and for students. In particular, a ranking of the students sorted by the number of problems solved is given. The system also includes tools to send comments about the problem, and ask questions to the teachers.

Compared to other disciplines, judging the correction of a program, with a high degree of certain, is relatively simple; that is what makes automatic evaluation feasible. However, there are many aspects of programming that are not so easy to evaluate: computational complexity, design and organization of the code, robustness, legibility, etc. In consequence, the task of human teachers remains to be essential.

3.3 A Judge-Based Methodology

As mentioned in section 2, on-line judging systems have already been applied in computer programming courses. We can distinguish four different kinds of activities using automatic judging:

- **Independent problems.** In this kind of activity, many problems are proposed to the students. The problems are independent of each other, and with different levels of difficulty. The students are expected to select and solve some of them, not necessarily all. Problems can be grouped by category, in such a way that each category illustrates a programming technique discussed in class. Typically, some weeks are given to complete the activity.
- **Dependent problems.** This case is preferable when the objective of an activity is to develop a longer and more complex programming project. The project is divided into small and consecutive subproblems; each of them is described as a problem in the judge. The students have to solve *all* the problems in the given order. In the itinerary, a number of programming techniques can be illustrated. This activity can normally take several months.
- **Contest-style.** Contrary to the other cases, here the presence of the students is required. A set of at most 9 problems is given to the participants. They have to try to solve as many problems as possible, and as fast as they can. The contest can take between 4 and 6 hours.
- **Designing problems.** This is the most creative type of activities. The students have to create a problem with the format of the judge: problem description, source code to solve it, input cases, and expected outputs. The ability of the students to produce original and relevant problems is evaluated.

4 Evaluation of the Method

In this section, we provide detailed information about the course where the experiment was carried out, the application of the methodology, and the obtained results. The study was conducted at the Computer Science Faculty of the University of Murcia (Spain). This Faculty has a long tradition –25 years– in teaching Computer Science degrees.

4.1 Participants and Background

Table 1 summarizes the main information regarding the course and degrees under study. In particular, the new methodology was first applied in the academic year spanning from fall of 2007 to spring of 2008. Considering the enrollments, two of the degrees (TECS and TECM) can be considered as highly populous, while the other (CSE) is a reduced group.

Table 1. Course and degrees where the new methodology was applied. "ECTS": equivalent load in European Credits Transfer System; "Enroll07": enrollment the year before the experience; "Enroll08": enrollment the year of the experience. CSE is a five-years degree, while TECS and TECM are three-years degrees.

Course name	**Year**	**Duration**	**ECTS**
Algorithms and data structures (ADS)	2nd year	Annual	12
Degrees	**Acronym**	**Enroll07**	**Enroll08**
Computer science engineering	CSE	56	44
Technical engineering in computer systems	TECS	162	162
Technical engineering in computer management	TECM	150	131
	Total	368	337

ADS is basically a course in programming, emphasizing issues of algorithms and data representation. This course introduces such topics as data structures –including hash tables, trees and graphs–, objects, abstract data types and formal specifications. The course also includes techniques for performance analysis and design of algorithms. Design techniques such as divide and conquer, the greedy approach, dynamic programming, backtracking, and branch and bound are presented through a variety of algebraic, graph, and optimization problems. The programming languages used to illustrate these concepts are C, C++ and the formal specifications language Maude [12].

ADS was traditionally organized in a *monolithic* form: weekly lectures, laboratory sessions, a final exam and a programming project for each semester. In fall of 2007, the three courses were involved in the new system. Even though the students were also given the possibility to follow the traditional method, more than 2/3 of them actively participated in the proposed activities.

4.2 Instantiation of the Method

Figure 2 shows a global view of the new organization of the course (right), as compared to its traditional design (left). Activities U2, U3 and U5 are partial-exams of the corresponding theory units; P2 is lab assignment involving a theoretical/experimental analysis of efficiency of the project created in P1. The remaining activities are programming contests done in Mooshak.

First Semester: Data Structures

Traditional organization	Continuous evaluation
Programming project + Final exam	U1. Formal specifications
	U2,U3. Hash and trees exam
	U4. Graphs
	P1. Data structures implementation

Second Semester: Algorithms

Traditional organization	Continuous evaluation
Programming project + Final exam	P2. Experimental analysis
	U5. Algorithmic analysis exam
	U6. Divide and conquer
	U7. Greedy algorithms
	U8. Dynamic programming
	U9. Backtracking
	U10. Branch and bound

Fig. 2. Comparison between the traditional methodology based on a "final exam" (FE) evaluation, and the proposed "continuous evaluation" (CE) methodology. The activities that use Mooshak are marked in gray. Activities U1, U2, etc. correspond to units of theory; activities P1 and P2 correspond to practice.

There is a great variety in Mooshak's activities: some of them are to be done in groups, and others individually; sometimes the problems are assigned to the students, other times they can freely choose; they can be dependent or independent problems, etc. Figure 3 provides more information.

When introducing the on-line judge in the course, most work is not done in the presence of the teacher. One of our main concerns was to guarantee the originality and authorship of the programs submitted by the students. Some strategies are applied to reduce the risk of plagiarism and to detect it:

- Some of the activities (P1 and P2) include a compulsory interview of each group with the teachers, where they have to demonstrate their authorship.
- Authorship is guarantied in partial-exam activities (U2, U3 and U5).
- For the activities done in Mooshak, we use a plagiarism detection system developed by Cebrian *et al.* [8]. Thanks to Mooshak, all the submissions are available in judge's server, so the plagiarism detector can be easily applied.

Finally, we have to note that all the activities have to be documented by the students (written by hand), and they are manually corrected by the teachers.

Activity	Type of problems	# prob. to pass	# prob. to max.	Total # prob.	Indiv./ group	Language	Assigned
U1. Formal specifications	Independent	14	23	26	Group	Maude	No
U4. Graphs	Independent	4	6	15	Indiv.	C/C++	No
P1. Implem. of data struct.	Dependent	13	16	17	Group	C++	No
U6. Divide and conquer	Dependent	1	2	16	Indiv.	C/C++	Yes
U7. Greedy algorithms	Independent	1	4	7	Indiv.	C/C++	Yes/No
U8. Dynamic program.	Independent	1	4	7	Indiv.	C/C++	Yes/No
U9. Backtracking	Independent	1	3	12	Indiv.	C/C++	Yes/No
U10. Branch and bound	Independent	0	2	12	Indiv.	C/C++	No
Local program. contest	Contest	0	3	8	Group	Java/C/C++	No
Total		**35 (8)**	**63**	**120**			

Fig. 3. Description of the activities in the CE method that use Mooshak. "# prob. to pass": minimum number of problems necessary to pass the activity; "# prob. to max.": number of problems to obtain the maximum mark; "Total # prob.": number of problems existing in the judge; "assigned": indicates if the students are assigned different problems or they can select by themselves ("yes/no" means some of them are assigned). Some activities should be done individually, and others in groups of two. All the activities were compulsory, except the last two that were optional.

4.3 Results of the On-line Judge

The overall impact of on-line judging in the teaching of ADS has been dramatic. Up to 273 of the 337 enrolled students (81%) participated in some activity of the judge; 268 of them (79.5%) solved at least one problem. This percentage raises up to 84% in CSE and TECS groups.

In total, the on-line judge received 16054 submissions: 11969 C/C++ programs and 4085 Maude programs in U1. This makes an average of 59 submissions per student: 44 C/C++ programs, and 15 Maude programs. The on-line judge classified around 6427 of these as correct (40.1%), and 4417 as "wrong answer" (27.5%). More information on the classification of the submissions, and the percentages per unit of knowledge is shown in Table 2 and in Figure 4.

Table 2. Detail of the classification of the submissions by activity, as listed in Figure 3. The last column indicates the number of students (or groups) that passed the corresponding activity.

Activity	**Total submissions**	**Correct**	**Wrong answer**	**Runtime error**	**Other errors**	**Pass the activity**
U1	4085 (25%)	1971 (48%)	1511 (37%)	600 (15%)	3 (0.1%)	85 groups
U4	2884 (18%)	1164 (40%)	493 (17%)	366 (13%)	861 (30%)	181 (54%)
P1	3615 (22%)	1229 (34%)	1099 (30%)	273 (8%)	1014 (28%)	41 groups
U6	2032 (12%)	705 (35%)	263 (13%)	161 (8%)	903 (44%)	178 (53%)
U7	1780 (11%)	619 (35%)	688 (39%)	66 (4%)	407 (23%)	182 (54%)
U8	830 (5%)	410 (49%)	189 (23%)	51 (6%)	180 (22%)	170 (50%)
U9	778 (4%)	309 (40%)	171 (22%)	12 (2%)	286 (37%)	162 (48%)
U10	50 (0.3%)	20 (40%)	3 (6%)	3 (6%)	24 (28%)	13 (4%)
Total	16054	6427 (40%)	4417 (28%)	1532 (10%)	3678 (23%)	75 (22%)

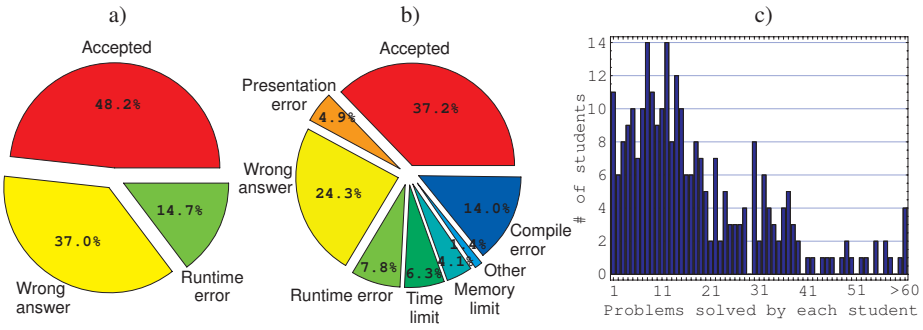

Fig. 4. Some results of the submissions made to the on-line judge. a) Classification of the 4085 Maude programs in U1. b) Classification of the 11969 C/C++ programs in the rest of activities. c) Number of problems solved by each student.

The average number of submissions per student until getting a program accepted is 2.7; anyway, many students found the solution to the problems at the first attempt (mode is 1). As shown in Figure 4, compilation errors, runtime errors, and excessive consumption of time and memory are caught by the system.

There is also a significant difference in the three groups under study. For example, while the average acceptation rate in CSE is 50%, in TECS it is 37% and in TECM 36%; this difference is consistently seen in all the activities.

Figure 4c) shows a histogram of the number of problems solved per student. This value covers a wide range, from 1 to 80, with an average of 18.5, standard deviation of 14.7, and with two modes of 8 and 12. As indicated in Table 2, many students passed individual activities –normally over 50%–, although not all of them passed all the activities.

Finally, it is also interesting to analyze *when* the students work. Figure 5 shows two charts representing the number of submissions per hour and per day of the week. The greatest period of activity takes place at lecture hours and days. However, submissions done by the students *outside* lecture hours represent a total of 48.6%, thus demonstrating the importance of autonomous work. In fact, there is no single hour with 0 submissions.

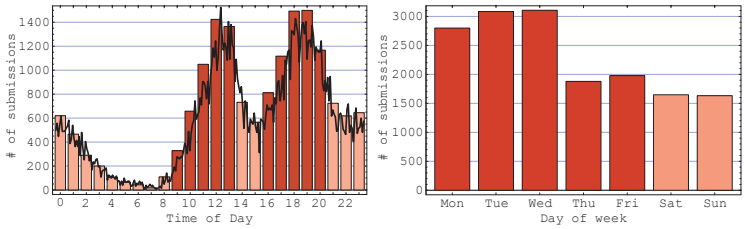

Fig. 5. Left: number of submissions made by the students in each hour of a day. The minimum is at 7 am (18) and the local maxima at 12 pm (1423) and at 7 pm (1499). In thick line, detailed number of submissions per 1/10 of an hour. Right: number of submissions in each day of a week. Darker bars represent lecture hours and days.

Concerning the days of the week, the distribution is quite surprising, with a peak on Wednesdays and a sudden decrease on Thursdays. Working habits also differ from day to day. On Wednesdays, the maximum activity is clearly located at 12 pm. However, in the rest of days the peaks are registered in the interval 6 pm – 7 pm; only on weekends most work is done between 8 pm and 9 pm.

5 Discussion

The results obtained after the application of the edutainment-based methodology are very promising. Considering just the final marks of the students, shown in Table 3, the new approach achieved excellent improvements. The most striking fact is the dramatic decline in the dropout rate, from 72.3% to 44.8%. On the other hand, the increase in the failure rate is due to the high number of students that passed some activities but not all. However, we think they can be better prepared to pass the course in subsequent attempts.

It is also observed that the improvement is far more significant in CSE than in TECS and TECM; the pass rate in CSE degree grows from 19.64% to 61.36%. The reduced size of this group has allowed a better application of the new methodology. It could also be due to a general *higher level* of students in this five-years degree as compared to the students in the "technical engineering".

Analyzing these results, we think the proposed organization of the course successfully meets all the pedagogical principles established in subsection 3.1:

- **Motivation.** In all the ways of using the on-line judge (independent, dependent and contest style problems), the ranking plays a fundamental role in motivating students to solve more problems, faster and more efficiently. If students get an "accepted", a rise in ranking means students get an incentive to continue tackling other algorithmic problems. This is evident in the high number of students who solved more problems than those necessary to obtain the maximum mark: they also wanted to be up in the ranking.
- **Active learning.** When students solve the proposed problems, they play a leading role in their own learning. As an example, it is worth underlining the high number of comments and questions asked by the students in Mooshak. In particular, 1119 questions were answered by the teachers along the course.

Table 3. Pass, failure and dropout rates of the three degrees where the new methodology was applied (in 2008), as compared to the results in the previous year (2007)

Final	**Total**		**CSE**		**TECS**		**TECM**	
results	2007	2008	2007	2008	2007	2008	2007	2008
Pass rate	10.9%	22.3%	19.6%	61.4%	7.3%	13.0%	11.1%	19.1%
	(40)	(75)	(11)	(27)	(11)	(17)	(18)	(31)
Failure rate	16.8%	32.9%	14.3%	6.8%	22.7%	32.1%	12.3%	40.7%
	(62)	(111)	(8)	(3)	(34)	(42)	(20)	(66)
Dropout rate	72.3%	44.8%	66.1%	31.2%	70.0%	54.9%	76.5%	40.1%
	(266)	(151)	(37)	(14)	(105)	(72)	(124)	(65)

- **Autonomous work.** Students can work in the labs, where they have help from the teachers. However, it is evident that students mostly work at home, and ask questions to the teachers by using Mooshak. Therefore, on-line judging eases to work autonomously.
- **Feedback of the learning process.** The web system provided feedback to help students to correct many errors of the programs, thus avoiding assistants spend much effort figuring out the causes of the failure, as happens in a traditional evaluation. The judge is accessible 24-hours a day and the feedback is instantaneous. From the point of view of the teachers, information is also comprehensive and immediate; they can analyze the difficulty of the problems, the evolution of the students, identify the best students, etc.

To conclude, we have to mention some remarkable results from a survey carried out among the students of the CSE group, regarding their experience in the new methodology. They were asked to indicate a degree of agreement/disagreement with a series of statements. These are some of the items evaluated:

- 77% of the students agree that "they learn better with the new method";
- 68% say that "the public ranking of the judge fosters competitiveness";
- All students disagree with "cheating is easier with the on-line judge";
- 91% say that "if they could choose, they would follow again the CE method".

These ratios have to be considered as subjective *appreciations* made by the students. Nevertheless, we have to observe that 75% of the students enrolled in 2008 were also enrolled in 2007. Thus, they were able to compare the new and the old methodology. Proving that they really *learn better* is difficult, as they were evaluated with different methods. However, considering only the theory units corresponding to U2, U3 and U5 (that were evaluated with a written exam, both in 2008 and 2007), the pass rate went from 18.6% in 2007 to 38.6% in 2008, and the average mark (in range 0-10) went from 4.6 to 5.7.

On the negative side, we can mention two items:

- 86% of the students agree that "the feedback provided by the judge in case of error is insufficient". In order to reduce this problem, we took the strategy of *releasing* one of the secret input/output cases for each problem.
- 60% say that "the total load of work is higher with the new methodology"; only 27% say it is lower. We think this problem can be easily solved by redesigning some of the activities. Nevertheless, the new methodology has a crucial advantage over the traditional one: the students work along the course, and not just some weeks before the final exam.

6 Conclusions and Future Work

In this paper, an innovative experience on computer science education –applying the techniques, concepts and methods of edutainment– has been presented. Entertainment has traditionally been neglected in formal education, and specially in university studies. We have shown that on-line judging systems can be used to make more fun the activities of a programming course.

In general, the results of our experiment are outstanding. The approach improves self-assessment skills and encourages students to work independently. The public ranking and other statistical data provided by Mooshak, promote competitiveness and offer appealing material to the students. The assessment is fair and objective, and students gain additional feedback from the human judges. Obviously, the improvements are also due to the introduction of continuous evaluation and the elimination of the final exam, which constitutes the second key element of the proposed methodology.

Two major aspects remain to be improved in the future: the feedback provided by the judge, and plagiarism detection. We are currently working on extensions of Mooshak to provide detailed feedback in case of "wrong answer", and to integrate the plagiarism detector subsystem into the web interface of Mooshak.

References

1. Education at a Glance: OECD Indicators (2006), `http://www.oecd.org/edu/eag2006`
2. OECD Science, Technology and Industry Scoreboard (2007), `http://www.oecd.org/sti/scoreboard`
3. Rapeepisarn, K., Wong, K.W., Fung, C.C., Depickere, A.: Similarities and Differences Between "Learn Through Play" and "Edutainment". In: Proc. of the 3rd Australasian Conference on interactive Entertainment, pp. 28–32 (2006)
4. Leal, J.P., Silva, F.M.A.: Mooshak: a Web-based, Multi-site, Programming Contest System. Software-Practice, and Experience 33(6), 567–581 (2003)
5. Robins, A., Rountree, J., Rountree, N.: Learning and Teaching Programming: A Review and Discussion. Computer Science Education 13(2), 137–172 (2003)
6. Guerreiro, P., Georgouli, K.: Enhancing Elementary Programming Courses Using E-learning with a Competitive Attitude. Int. Journal of Internet Education (2008)
7. Guerreiro, P., Georgouli, K.: Combating Anonymousness in Populous CS1 and CS2 Courses. In: Proc. ITICSE 2006, pp. 8–12 (2006)
8. Cebrian, M., Alfonseca, M., Ortega, A.: Towards the Validation of Plagiarism Detection Tools by Means of Grammar Evolution. IEEE Trans. on Evolutionary Computation (2008)
9. Khine, M., Suja'ee, M.: Core Attributes of Interactive Computer Games and Adaptive Use for Edutainment. In: Pan, Z., Cheok, D.A.D., Müller, W., El Rhalibi, A. (eds.) Transactions on Edutainment I. LNCS, vol. 5080, pp. 191–205. Springer, Heidelberg (2008)
10. White, R.: That's Edutainment. Hutchinson Leisure & Learning Group (2003), `http://www.whitehutchinson.com/leisure/articles/edutainment.shtml`
11. Martens, A., Diener, H., Malo, S.: Core Attributes of Interactive Computer Games and Adaptive Use for Edutainment. In: Pan, Z., Cheok, D.A.D., Müller, W., El Rhalibi, A. (eds.) Transactions on Edutainment I. LNCS, vol. 5080, pp. 172–190. Springer, Heidelberg (2008)
12. Clavel, M., Duran, F., Eker, S., Lincoln, P., Marti, N., Meseguer, J., Talcott, C.: All About Maude - A High-Performance Logical Framework. In: Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C. (eds.) All About Maude - A High-Performance Logical Framework. LNCS, vol. 4350. Springer, Heidelberg (2007)