Using Mooshak as a Competitive Learning Tool

José Paulo Leal and Fernando Silva

CRACS & DCC-FCUP, Universidade do Porto, Portugal
{zp,fds}@dcc.fc.up.pt

In this paper we present Mooshak, a system originally developed for managing programming contests, as a tool for competitive learning. We start by focusing on the features that enable or simplify this type of usage and we presented two concrete examples of successful use of this system in two courses with very different requirements.

1. Introduction

Mooshak [1] is a web based competitive learning system originally developed for managing programming contests over the Internet. From start, in 2001, the system followed the ICPC contest rules and has been used in several local and regional contests as well as for multi-site contests. The system has evolved to include features such as a more sophisticated automatic judging, secure and safe process execution environment, full accounting of memory usage, on-line registration, XML import/export for problem descriptions and users and virtual competitions. Virtual competitions are a wonderful tool for contestants to practice as if they were participating in a real contest, thus experiencing all the contest events in real time. Mooshak has also evolved to manage other types of contests, such as the Portuguese section of the International Olympiad in Informatics (IOI) and the Portuguese Logic and Functional Programming contests, with different rules and different grading policies.

From very early, Mooshak started being used as an e-learning tool in several universities specially in courses on programming, algorithms and data structures, artificial intelligence, among others. In these courses students participate in several "contests" where they have to solve one or more problems, receive immediate feedback on their attempts and are able to compare their own progress with that of their colleagues. These are characteristics of competitive learning that stimulate students to work harder on problem solving using the subjects taught in each course.

In this paper we present the main features of Mooshak and how it in can be used in competitive learning. We present two case studies that make different uses of Mooshak as a competitive learning tool, creating several programming "contests" with different rules. In one of the examples, students participate in several, usually five, assignment-contests, each with a week or ten days duration. Student's participation is remote (they don't have to be in a specific location) and evaluation follows a mixed strategy. Each student submission is automatically evaluated by Mooshak in the traditional black-box approach. This evaluation is later complemented by having the students presenting and explaining their solution to teaching instructors in order to receive feedback and credit for their work. In the other example, "contests" have a short duration (less than one hour), students participate in a specific and controlled location and during the contest students receive error messages explaining the detected semantic errors on their submissions. The purpose of this paper is not to compare the differences of the two approaches but rather to highlight what they have in common from a competitive learning point of view. To assess the methodology we used questionnaires to find evidence of the positive role of competitive learning in computer science education.

2. System overview

Mooshak was designed as a web application to manage and run ICPC programming contests. It manages all phases of a programming contest, from the setting up of a contest, to the activities after the contest, including the management of the contest itself. Mooshak provides different user interfaces targeted to different user profiles. Although these profiles reflect the roles of the different participants in a programming contest, they can be adapted to the typical roles of a Learning Management System (LMS). The four interface types provided by Mooshak are:

Administration view used by the contest-director/course-coordinator to setup a new contest/activity and to add all necessary data to make it operational;
Teams view used by contestants/students to communicate with the judges/teachers, asking questions, submitting programs, and requesting printouts;
Judges view used by judges/teachers to give feedback to contestants, by answering questions, marking programs and tracking the handling of printouts;
Public view allows any user on the Internet to follow online the progress of the contest/ course.

Access to these user-oriented views is controlled by authentication, except in the case of the public view that is open to everyone. Furthermore, the system has built-in safety measures to prevent users from interfering with the normal progress of the contests or pedagogical activities.

Mooshak was designed for managing several contests in a single installation. A typical event has more than one contest: the official contest, an training session. The administration interface was designed to allows contest directors to setup all necessary data to run the contest. By contest data we mean the problem set (problem descriptions and test cases), teams composition and authentication (passwords), and programming languages and corresponding execution commands and compilation flags. When used as pedagogical tool, a Mooshak installation is used for a course. Several installations of Mooshak for different courses can share a single server. The concept of contest is usually mapped into a course assignment. New features were added to Mooshak that simplify using contests as courses that are described in the following sub-sections.

2.1. Grading policies

By default, Mooshak follows ICPC rules regarding grading and classification. These rules are specific to ICPC contests and other types of contest, as the IOI where Mooshak has also been used, have a completely different set of rules. To cope with these differences Mooshak is deployed with several grading and classification policies and new policies are very easy to add: if a new package with a specific "interface" is dropped in the appropriate directory, a new policy is immediately available to any assignment/contest.

To support new grading policies, new fields were added to program descriptions. For instance, The program size in bytes is recorded to support a short program policy. There are contests where solving the problem is not enough for winning the contest, the code must also be the shortest. Other field where added for supporting grading policies that can be used in class. For instance, test cases can be assigned with points so that the final grade is computed by adding the points of each passed test.

Most of these policies are objective in the sense that they automatically compute a grade by processing the submitted program. By contrast, in a subjective policy (part of) the grade is filled in by the teacher. This type of policy is useful for grading features of the program's code that are not easy to grade automatically, such as programming style.

2.2. Special correctors

The standard way of evaluating a program is to compile it and then execute it with a set of predefined input files. The program is accepted if compiles without errors and the output of each execution is exactly equal to the expected output. Special evaluators are custom programs that change this general evaluation pattern for a given programming problem. Mooshak allows two types of special evaluators: static correctors and dynamic correctors.

> **Static correctors** are invoked immediately after compilation, before any execution, and receive as input the code of the program being evaluated. They can be used in very different ways to: compute software metrics on the source code; perform unit testing on the program (skipping the execution part); to check the structure of the program's source code.
> **Dynamic correctors** are invoked after each execution with a test case. This type of corrector can be used for dealing with non-determinism. If the solution is a set of values that can be presented in any order then a dynamic corrector can be used to reduce the output to a normal form.

Special correctors are configured as command lines associated to the definition of a problem. These command lines can make use of a set of predefined variables to refer to relevant parts of the problem definition, such as the source code (for static correctors), or input and output files (for dynamic correctors). This data is also available to correctors as environment variables.

The exit status of static and dynamic correctors affects Mooshak's automatic evaluation procedure. Each classification has an associated numeric value - 0 for Accepted, 1 for Presentation Error, 2 for Wrong Answer and so forth - and the exit value of the corrector will set the classification of the submission in that stage.

2.3. Feedback to students

In ICPC contests teams receive very limited feedback when their submissions are rejected. For instance, teams know that they have a compile error but they do not know of that error was.This is reasonable in a competition where all teams have access in their workstations to the same compiler, with the correct compilation flags, the same used for judging their program. When students use Mooshak from different labs, their laptops or their home computers, they may use different compilers, or different versions of the same compiler.

To cope with this problem Mooshak can be configured to show details of every type of error, not only compilation errors. For instance, Mooshak can be configured to show execution errors and, in a Java program, students receive a detailed error message.

2.4. Authentication

When managing a course the teacher will upload students data from the faculty information system. Then, (s)he can generate passwords for each student, as would be done for a contest but this procedure is rather cumbersome, as students tend to forget passwords. An alternative is to use existing directory services for authentication, such as LDAP. With this approach the students use the same passwords they have for other services, such as lab accounts.

Alternatively, some course coordinators do not create student accounts in first place but configure the contest/assignment to allow user registration. In this case the student has access to a form for registration and receive a password by email. When using this method, course coordinators usually ask students to use a conventional ID to simplify their identification.

3. Handling Programming Assignments with Mooshak

For several years, Mooshak has been used as an e-learning environment to manage programming assignments in several computer science courses. In one such course, a typical data structures and algorithms course taught in the third semester of our undergraduate program in computer science, students were given four or five programming assignments during the semester. Each assignment involved devising a solution program that solves a problem described in a very similar way to programming contests problems. The assignments typically covered the following themes:

- Linked lists, stacks and queues
- Binary trees and heaps
- String pattern matching, hashing and sorting
- Graphs
- Labyrinths

Solution programs had to be written in Java and the students could not make use of the Java classes that were directly related with the theme of the assignment. For example, if the theme was "linked lists, stacks and queues", students were not allowed to use classes such as LinkedList, Stack or Queue from Java. They had to write their own implementation of those classes, at least once, so that they would gain a better understanding on the implementation of fundamental basic data structures. However, for the assignments that followed the students were encouraged to make use of the Java classes that they couldn't use in the previous assignment. For example, if they needed to use linked-lists, then they were encouraged to make use of the LinkedList Java utility class.

Given the size of the class, over 150 students, students were grouped in teams of size two, and given access to Mooshak. For each assignment students had at least one week to solve it and during that week they could ask for assistance from the teaching assistant of their class. Furthermore, during the week preceding the assignment, the students were encouraged to solve practice problems on the same theme.

The assignments were planned so that students had at least a week to think, program and submit a solution to Mooshak. The system provides immediate feedback on their solution and students can submit any number of times, even after having a solution accepted. Usually, students are given two instances of the same problem in Mooshak. One instance uses simpler test cases, thus not requiring very efficient solutions. Another instance of the problem had more complex test cases to test students' solutions for efficiency both in terms of memory and execution time.

Since students can submit their solutions from anywhere, we introduced a further requirement that is mandatory in order for students to receive marks for their solution. This involved having each team to present their solution to the teaching instructors, in the class immediately following the submission deadline. This introduces some control over the whole process, thus allowing us to minimize fraud and also to provide feedback to students. The marking of each assignment takes into account the performance of the solution on Mooshak, that is whether it was accepted with both the simpler and the more complex test cases, the team's presentation of their solution, the quality of the code they wrote and the algorithm used to solve the problem.

Having run this course for the last five years using Mooshak, some advantages can be easily identified:

- Students do make an effort to understand the materials taught in classes about fundamental data structures and algorithms.
- Students exercise their programming skills as they must implement a reasonable number of problems.
- Students learn how to do team work, even though this is not a fundamental issue for the course.
- Students are able to monitor their progress by submitting their solutions and receiving immediate feedback.

- Students learn to think on test cases in order to discover bugs in their solutions.
- Teachers can automate assignment evaluation and thus handle larger classes.

The experience also showed disadvantages of this approach, some of which can easily be handled with. Students feedback, very often point out their disappointment for not receiving more helpful evaluation messages whenever they submit a solution that fails on Mooshak. This is something we noted can cause some stress to students, sometimes leading the students to drop out on the assignment. A solution to this negative assessment from students could be to prepare Mooshak to provide guidance to students regarding the reasons that lead their solution to failure. For example, whenever a student submits a solution to a problem, if it doesn't pass all test cases, it should report to the student the input test case which lead to failure. In showing the student the test case, allows him to go back to his code and rethink what he is doing wrong, solve it and submit the new solution again. This feedback provided to students can, in our view, have a better effect on the students learning outcomes then the feedback a teaching assistant could offer. Rather then being told on what's wrong in their program, the students have to find out by themselves by looking at the expect input and output, thus appealing to the students for an in depth understanding of the solution he his devising.

Our experience has also shown that this approach is rather demanding from students. They are confronted with new challenges every other week. This pace isn't easy for many students and we feel we needed more time to coach the students. This is more noticeable with students whose programming preparation is not yet as it should be to attend this course. However, in general terms our experience has been very rewarding and we noticed that the students that approached the practicals seriously and did well on the assignments usually do also very well on the course.

4. Custom automatic evaluation

The automatic evaluation of Mooshak, designed for programming competitions, is adequate to handle programming assignments in courses such as data structures and algorithms. The solutions to these assignments are programs that process a data stream and produce a data stream. This may seam a fairly good model for most programs but does not fit all cases. For instance, many software components interact with a complex environment through an application programming interface (API) and react to events instead of processing the input. This kind of program requires a different type of automatic evaluation since they have different execution model. On the other hand, evaluating the execution is not the only way to evaluate the program. In some cases in may be preferable to evaluate only the source code, skipping the execution altogether.

We had a chance to experiment with other types of automatic evaluation in a course on human computer interaction. The assignments given in this course required the development of a graphical interfaces (GUI) using different frameworks, such as HTML and Java GUI toolkits. In the HTML assignment students have to code a simple HTML page containing a form with objects positioned using tables. In the Java toolkit assignment students have to code a simple AWT interface with basic widgets (labels, buttons, editable fields) using standard layout managers. In both cases we used static correctors to evaluate student programs. These evaluators are invoked after compilation and in both cases they replace the default dynamic evaluation of Mooshak; that is, no execution of the program is performed afterwards. Both evaluators have in common the fact that they are based on matching structures [2]; nevertheless they significantly different. These evaluators will are included in Mooshak distribution as contributed code.

4.1. HTML evaluator

The HTML evaluator is strictly a static evaluator. Since HTML is not actually "compiled" and "executed" but rendered on a screen, it does not make sense to evaluate it using a traditional approach. Instead, the HTML static evaluator compares the source code produced by the student with the source code of the solution. Both HTML codes are parsed into an hierarchical data structure and then compared. Comparing two HTML data structures follows a straightforward algorithm, similar to pattern matching: the same elements must have the same descendants in the same positions and have the same attributes, and texts must be equal. Grading is equally straightforward: missing and additional attributes, elements and text have an associated penalty.

The main drawback of the HTML evaluator is the fact that it only accepts submissions that are very similar to the solution. On the other hand, it produces very accurate feedback that help the student to correct their mistakes. In HTML assignments we use a classification policy that does not penalize repeated submissions. Hence, students can make multiple submissions until they have the problem accepted or until they run out of time.

4,2 AWT evaluator

The AWT evaluator is not strictly a static evaluator. It compiles the Java code and executes the compiled code, but it does not feed executions with data through streams, as in the default evaluator of Mooshak. Instead, it passes events to certain widgets and checks if the changes in the GUI correspond to what was expected. This evaluator works in two parts: 1) it compares the GUI of the student's program with the GUI of the solution; 2) if the GUIs match then it performs unit tests on the GUI of the student's program.

To match the two GUIs we could have used an approach similar to the one used on the HTML static evaluator. A program with a GUI creates a hierarchy of graphical components, with widgets (buttons, labels, etc) and containers. Nevertheless, to use the same approach we would have to restrict the type of layout manager used for placing the descendants of each container. Otherwise, the two GUIs could be identical and have a different structure. Instead, we decided to use a different approach: the matching algorithm ignores containers and matches widgets the occupy roughly the same relative position. For instance, if the solution has a widget of type Button on the right lower corner then it will match with a GUI with a widget of the same type in the same position. The class hierarchies of both widgets are compared for a non-trivial common type. For instance, the student can extend the class Button and still this extended class would match a button in the solution program. On the other hand, all widgets share a common Component class and trivially match if this type was not ruled out.

When two widgets of same type and relative position are found, the matching algorithm proceeds by checking the widget properties. For instance, two buttons must also have the same label. Not all properties are relevant for this type of matching. The GUI matcher was a configuration file stating, for each type, the properties that are relevant for matching. For instance, a label is relevant property of class Button, but since class Button a specialization of Component, then the font property from this class must be checked too.

As in the HTML static evaluator, the matching of Java GUIs is also used for grading and feedback. If a property has a wrong value then this is reported to the student and a penalty is added to the grade. It should be noted that the student can resubmit his program within a certain time limit, and have a chance to fix these mistakes without negative repercussions in their final grade.

5. Conclusion and future work

In this paper we presented Mooshak, a system originally developed for managing programming contests following ICPC rules, and we showed how it can be used as tool for competitive learning. We focused on the features that were introduced in Mooshak to enable or simplify its use as a competitive learning tool and we presented two concrete examples of successful use of this system: a course on data structures and algorithms with programming assignments following the ICPC problem description and evaluation style, and a course on human-computer interaction that required custom evaluators.

We think that these examples show the effectiveness of Mooshak as a competitive learning tool but our personal experience, combined with feedback from other users of Mooshak as an e-learning tool, reveal that system lacks a few features that we expect to implement in the near future.

Feedback to students - Mooshak already has some support for feedback. However, the available feedback is not enough as some students may get stuck on a particular problem and sometimes give up on solving it. We are considering to add feedback messages to tests in order to give hints to students. Some test cases check a particular situation (negative values, empty sets, input of large size) and this information could help students overcome the problem. In some circumstances, showing the test case can help to overcome the problem at it may also be a possibility to consider.

Automated reevaluation - The reevaluation is a feature available since the first versions of Mooshak. In a contest it is sometimes necessary to reevaluate submissions: if an error is detected in a test case; or if the evaluation process was not completed, due to the excessive load of the server, for instance. When used on a regular basis as a pedagogical tool, the chances for these kind of problem increase significantly. A recurrent demand from Mooshak 's users is the possibility of automating the reevaluation of the submissions with similar characteristics. For instance, all submissions in a certain programming language, to a certain problem, from a certain student, between certain dates, with a certain classification, and any combinations of all the previous.

Plagiarism detection - One of the major source of problems with e-learning systems is the increase in plagiarism. Although we tried to cope with it by requiring students to discuss their assignments, it would be helpful if the system could automatically detect signs of plagiarism. To be sure, it is possible to use existing plagiarism detection services to check student submissions to Mooshak but the procedure is rather cumbersome.

Bibliography

[1] José Paulo Leal, Fernando Silva "Mooshak: a Web-based multi-site programming contest system", Software: Practice and Experience, May 2003, vol. 33 n.6 (pp 567-581).

[2] José Paulo Leal, Nelma Moreira, "Using matching for automatic assessment in computer science learning environments", Web-based Learning Environments, June 2000.