World Transactions on Engineering and Technology Education *© 2013 WIETE*
Vol.11, No.4, 2013

Introductory programming course: motivating students with prior knowledge

Luka Fürst & Viljan Mahnič

University of Ljubljana
Ljubljana, Slovenia

ABSTRACT: In this article, the authors present an approach to motivating the students of an introductory programming course at the University of Ljubljana, Ljubljana, Slovenia. They focus on the students whose background knowledge of programming is at least equivalent to the level taught in the course and who, consequently, tend to feel disappointed with the course. To make the course more attractive to experienced programmers, the authors introduced two annual events: a two-player board game programming competition and a contest for project proposals on the subject of inheritance in object-oriented programming. Both events regularly attract considerable interest from all students attending the course, not only from those with sufficient prior knowledge. The contest for project proposals benefits not only the proponents themselves but also the teaching staff, since the best proposals may be re-used as homework projects in the future years.

INTRODUCTION

University-level introductory programming courses are often attended by students with diverse backgrounds [1-3]. By noting the differences between the classes of 1985 and 1999 at the University of Leeds, Jenkins and Davy argue that the diversity tended to increase over time [3]. In the 1970s and 1980s, when computer usage was still far from widespread, computer science departments were populated mostly by enthusiasts and students with a strong interest in mathematics or engineering. Today, however, computers are omnipresent, and computer science is often viewed as a field that promises future employment. Consequently, the student body has become highly diverse in terms of both background knowledge and enthusiasm.

In the Faculty of Computer and Information Science (UL-FCIS) at the University of Ljubljana, Ljubljana, Slovenia, a similar trend has been observed. In addition to the reasons stated in the preceding paragraph, there are many differences in the formal education of students prior to entering university. In Slovenia, there are several types of high schools (secondary schools attended by students of age 14-18) with significant variety in curricula. Students coming from general high schools (called *gymnasiums* in Slovenia and many parts of central Europe) often have a solid background in mathematics, but many of them possess little or no programming knowledge. Students coming from engineering high schools are, on average, more skilled in programming, but often lack deeper mathematical and theoretical knowledge. Students who attended other high schools may lack both programming skills and mathematical knowledge.

Roughly half of the students enrolling each year appear to be complete programming beginners. On the other hand, some first-year students had already gathered respectable programming experience prior to entering university. Approximately every other year, UL-FCIS has a former IOI (International Olympiad in Informatics) medallist. Needless to say, the programming knowledge of such students substantially exceeds that which is required by the introductory programming course at UL-FCIS, at least in the area of procedural programming. Many experienced programmers have thus been disappointed with at least part of the course; their boredom during the compulsory laboratory sessions could often be observed. It has become clear that such students need additional motivation.

The first thoughts about motivating experienced programmers came into being in 2008/09, when the authors launched a competition of programs for the *Connect Four* board game. (In Slovenia, the academic year runs from 1 October to 30 September.) Since then, board game programming competitions have been held every year. At the beginning of each year, the students receive a framework that provides graphical user interface, deals with human-computer interaction, enables human-*versus*-human, human-*versus*-computer, and computer-*versus*-computer play, and imposes time controls. The task of the prospective participants is to write their own move selection logic (the engine). The competition is potentially attractive to the entire student body, since only basic programming knowledge is required to create a working engine. To reach a high rank, however, prior experience with advanced algorithms is definitely advantageous. The best participants are awarded bonus points in the final written examination.

A second idea was to introduce a contest in preparing homework project proposals on the subject of inheritance in object-oriented programming. This contest is associated with the last compulsory homework project, the subject of which is inheritance. Instead of completing the project prepared by the teaching staff, every student can choose to prepare his/her own project proposal together with a solution to it. Such a contest has been held every year since 2010/2011. Since meaningful and original inheritance-based programming problems tend to be difficult to find, the participants in the contest benefit themselves but, potentially, also the teaching staff. As with the board game programming competition, bonus examination points are awarded to the best participants.

Both types of events have attracted considerable interest, and both have enlivened the introductory programming course. Many students, experienced programmers and beginners alike, have participated in one or the other event; some of them have even been engaged in both. By now, both the programming competition and the project proposal contest have become a well-established part of the course.

Programming competitions appear to be a fairly popular idea to animate programming courses around the world. Ladd and Harcourt [4] for example, present a robotic game competition within an introductory programming course. Games and multimedia are another common way to motivate students [5][6]. Project proposal contests within introductory programming courses - in other words, the participation of students in *creating* programming tasks rather than just solving them - are probably much less widespread, at least when judging by the number of papers published. Djordjevic reports on simple game project proposals motivated by student interests, although it is not clear whether the proposals were collected in the form of a contest with strictly defined rules, as is the case with the course at UL-FCIS [7].

In the rest of this article, the authors first present the introductory programming course at UL-FCIS, followed by a description of the board game programming competition and the contest for project proposals on inheritance.

THE INTRODUCTORY PROGRAMMING COURSE AT UL-FCIS

The introductory programming course at UL-FCIS takes place in the first semester (approximately from 1 October to 20 January) of the first year [8]. The course teaches the basics of procedural and object-oriented programming in Java, the language of choice at many universities throughout the world [9-11]. The syllabus of the course covers control structures, arrays, classes and objects, inheritance and fundamentals of computer graphics. The course comprises 15 weeks of formal lectures and 13 weeks of laboratory sessions. Every year, approximately 250 students attend the introductory programming course. The lectures are attended by all students at once. The laboratory sessions are conducted in 9-10 groups comprising 25-30 students each. Formally, each group comprises about 15 students, but every laboratory session block (two consecutive academic hours) is attended by two groups simultaneously and supervised by two teaching assistants.

The students’ obligations for the introductory programming course at UL-FCIS comprise three homework programming projects, two theoretical tests and a final written examination. Each homework project and each test is worth 10 points, and the final examination is worth 60 points, giving 110 points in total. To complete the requirements of the course, the students have to amass at least 50 points, 25 of which have to be obtained in the final examination.

The subjects of the homework projects are essential control structures (first project), arrays (second project) and object-oriented programming with inheritance (third project). Homework projects are not graded directly but via a set of *upgrades*. The students have to implement three increasingly difficult upgrades of the project. The homework project alone is worth four (4) points, and the upgrades are worth two (2) points each. The students work on the upgrades in their groups during three dedicated weeks of laboratory sessions. The first project is graded at the beginning of November, the second one at the beginning of December and the third one at the beginning of January.

Since the laboratory session blocks are scattered throughout the week, there is a high risk of plagiarism between different groups. To reduce potential plagiarism, a separate set of upgrades is prepared for each group. Since it is hard to devise 9-10 different sets of upgrades for the same project, four to five (4-5) different projects per subject are prepared so that the number of upgrade sets per project diminishes to two or three.

BOARD GAME PROGRAMMING COMPETITIONS

The first idea to motivate students with advanced programming skills was to launch a two-player board game programming competition. Every year since 2008/2009, the students have been invited to write their own engine for the game selected for that year’s competition. In 2008/2009 and 2010/2011, the students’ engines competed in Connect Four. In 2009/2010, the game chosen was Battleship. The 2011/2012 and 2012/2013 competitions featured Dots and Boxes and Nine Men’s Morris, respectively. In 2013/2014, the authors introduced Polite Queens, an original game inspired by the well-known eight-queens puzzle. The two players alternately place queens on the squares of an $N \times N$ chess board. The player on move may place a queen on any free square that does not lie in the same row, column or diagonal as some already placed queen, including the queens placed by the players themselves. The player who runs out of moves first loses.

At the beginning of each academic year, a framework is prepared that provides the entire game-playing functionality except for the move selection logic, which, of course, has to be prepared by every participant of the competition. The framework is a Java program that can run the game in the human-*versus*-human, human-*versus*-engine or engine-*versus*-engine mode. The students' engines have to be written as Java classes. The framework accepts several optional parameters: the identifiers of the class(es) implementing the engine(s) to be engaged in the game, the time limits for both players, the delay after each move in the engine-*versus*-engine mode, etc. During the game, the framework graphically displays its current state, as well as the identities and the remaining time for both players. In the engine-*versus*-engine mode, the framework operates without any human intervention. In the other two modes, the framework enables the user to enter individual moves using the mouse.

At the beginning of each game, the framework creates an object for each engine (a Java class instance) to be involved in the game. To communicate with an engine, the framework invokes one of the following polymorphic methods on the engine's object:

- *Void newGame ()*: The framework invokes this method whenever a new game begins. The engine is expected (but not required) to initialise its data structures;
- *Void acceptMove(Move move)*: This method is invoked whenever the engine's opponent (a human or another engine) makes a move. The engine is expected (but not required) to update its data structures;
- *Move makeMove(long timeLeft)*: This method is invoked whenever it is the engine's turn to make a move. The engine has to return a legal move in at most *timeLeft* milliseconds; otherwise, it loses immediately. The engine may not spend more than a fixed amount of time for the entire game;
- *Void endOfGame(int outcome)*: This method is invoked at the end of each game. The parameter outcome can be 1 (the engine has won), -1 (the engine has lost) or 0 (the game has been drawn).

The engine has to be defined as an implementation class of an interface included in the framework. The interface contains the declarations of the above four methods. By using polymorphism, the framework can communicate with the engine without knowing its implementation; the minimum common interface is sufficient. The *makeMove* method is executed in a separate thread so that the user interface remains responsive while waiting for the engine to select a move. Figure 1 and Figure 2 show running frameworks for the games Nine Men's Morris and Polite Queens, respectively, in the human-*versus*-engine playing mode.

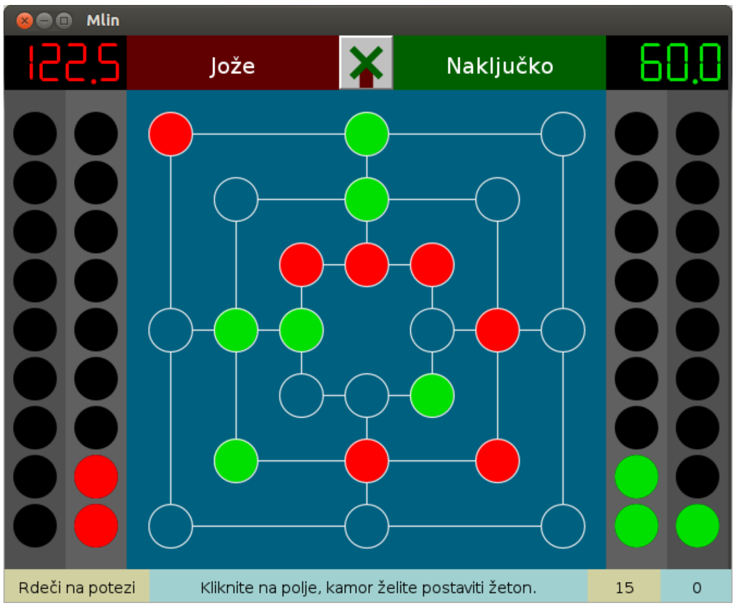

Figure 1: Nine Men's Morris.

To alleviate the efforts required to set up a working engine, the authors prepared a special engine that selects legal moves completely at random. This engine can serve as a useful *sparring partner* for the engines created by the students. Despite having no intelligence, the random engine is guaranteed to produce legal moves within the prescribed time limits, which is not always the case with the students' engines.

There are no conditions for entering the competition other than attending the introductory programming course at UL-FCIS. The knowledge that the students gain during the course completely suffices for writing an engine that plays in accordance with the rules of the game; all they have to do is to redefine the four methods given above. However, experience has shown that top-ranking competitors typically possess programming skills and knowledge exceeding the level of the introductory programming course, such as advanced recursive game-tree-search algorithms (e.g. Alpha-Beta [12]) coupled with heuristic time management strategies.

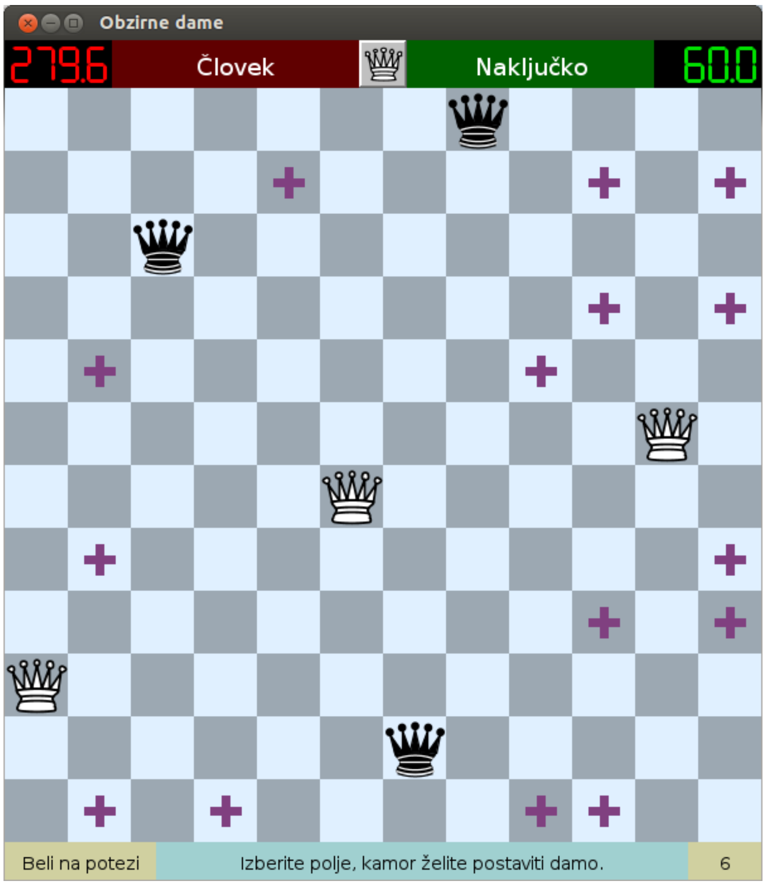

Figure 2: Polite queens.

The competition is divided into two stages. In the first stage, the participants' engines play against each other. Each engine plays a fixed even number of games against each of the other engines, half of them as the first player and half as the second player. In the second stage of the competition, the best five engines compete against their authors. Each engine plays two games (one as the first player and one as the second) against each of the authors of the other four engines.

The final rankings of the engines are determined by the sum of the score carried over from the first stage (8 points for the best engine, 6 for the second-best, 4 for the third-best, etc) and the score obtained against the authors (1 point for each win, half a point for each draw). The authors of the best ten engines (all engines participating in the second stage and the engines ranked 6-10 in the first stage) receive bonus points in the final examination. The winner is awarded 20 points.

By each of 2008/2009, 2009/2010 and 2010/2011, the competition attracted 12 students. In 2011/2012, as many as 20 students participated in the competition. Unfortunately, only three students competed in 2012/2013. The lack of interest was probably due to the relative difficulty of the Nine Men's Morris rules. At the time of writing, the 2013/2014 competition has not yet started. Since the rules of this year's game are significantly simpler than those of Nine Men's Morris, the number of participants is expected to reach at least pre-2011/2012 levels.

CONTESTS FOR PROJECT PROPOSALS ON INHERITANCE

Inheritance is a key concept in object-oriented programming and, hence, plays a major role in the syllabus of the introductory programming course at UL-FCIS. Inheritance has been the subject of the third homework programming project for many years. To prevent plagiarism between different student groups, four to five (4-5) different projects and different sets of upgrades for each project are prepared, so that two to three (2-3) groups share a project, but each group has its own set of upgrades.

The difficulty of preparing novel inheritance-based projects has been steadily increasing, since the number of meaningful real-world inheritance hierarchies appears to be limited or at least hard to find. For this reason, it was decided to harness the power of the students' imagination. Every year since 2010/2011, the authors have encouraged the students to propose their own projects on inheritance as an alternative to completing the third homework project. Student proposals have to meet the following criteria:

- The proposal has to contain the specification of the project itself, the specification of three increasingly difficult upgrades of the project and solutions to the project, and to each of the upgrades.
- The difficulty level of the project has to be appropriate for an average first-year student.
- The project specification has to require that a solution to the project contains:

  - at least three classes with inheritance relationships (both the $A \rightarrow B \rightarrow C$ and $A \rightarrow \{B, C\}$ types of hierarchy are possible);
  - at least one constructor per class;
  - at least one method to be defined (or declared as abstract) in a super class and redefined in subclasses;
  - a separate class for testing the inheritance hierarchy;
  - a polymorphic array of objects of the hierarchy.

- At least one of the upgrades has to require the introduction of an additional subclass and the redefinition of at least one method.
- Preferably, the project should address an interesting real-world problem.

Every proposal meeting the above requirements is treated as a complete 10-point substitution of the third homework project. The authors of the best five proposals receive bonus points in the final examination; the winner is awarded 10 bonus points.

The authors have received between 10 and 20 proposals each year. Some of them have been included into the database of projects on inheritance. One of the proposals, for example, features an imaginary construction company that builds houses and tower blocks. Both can be regarded as types of buildings, giving the hierarchy: Building → {House, TowerBlock}. The main objective of the project is to create a random array of different buildings, to calculate the price of each, and to produce a simple character-based visualisation of all buildings in the array. The price of a house has to be calculated in a different way from that of a tower block. Likewise, houses and tower blocks have to be visualised as different shapes. These stipulations imply that the methods for calculating the price of a given building and for producing a visualisation of a building should be (re)defined in both subclasses of the class Building. The objective of one of the upgrades is to add the class Dormitory as a subclass of Building and to redefine the method for calculating the price. This proposal perfectly satisfies all the requirements and has, therefore, become part of the database. Its author was pronounced the winner of that year's contest.

CONCLUSION

The authors have presented an approach to motivate introductory programming students with prior knowledge. In particular, they launched two types of events: a board game programming competition and a contest for homework project proposals on inheritance in object-oriented programming. Both types of events have caught the interest of many students, even the most experienced ones. In both 2010/11 and 2011/12, the programming competition was won by an IOI medallist. The database has been enriched by several original contributions from the project proposal contest.

The programming competition and the proposal contest have attracted not only highly experienced students. Several beginners have participated in the proposal contest and contributed original ideas. In board game programming contests, the number of beginners has been lower, since it is hard to reach a high rank without substantial experience.

With the exception of the year 2012/2013, the board game competition has always been attended by at least 12 students. The lack of interest in 2012/2013 can be at least partly explained by the relative difficulty of the Nine Men's Morris rules as compared with those of Connect Four or Dots and Boxes. The motivation in the form of bonus examination points apparently did not help much. For highly experienced students, the bonus points seem to be irrelevant, since most of them would receive the highest grade anyway; their motivation can be drawn only from the competition itself. For average students, though, the possibility of earning a few bonus points did not outweigh the perceived efforts required to tackle the complexity of Nine Men's Morris. To avoid repeating the same mistake, the rules of the 2013/2014 game are considerably simpler.

REFERENCES

1. Gill, T.G. and Holton, C.F., A self-paced introductory programming course. *J. of Infor. Technol. Educ.*, 5, 95-105 (2006).
2. Davis, H.C., Carr, L.A., Cooke, E.C. and White, S.A., Managing diversity: experiences in teaching programming principles. *Proc. 2nd LTSN-ICS Annual Conf.*, London, UK (2001).
3. Jenkins, T. and Davy, J., Dealing with diversity in introductory programming. *Proc. 8th Annual Conf. on the Teaching of Computing*, Edinburgh, UK, 81-87 (2000).
4. Ladd, B. and Harcourt, E., Student competitions and bots in an introductory programming course. *J. of Computing in Small Colleges*, 20, **5**, 274–284 (2005).
5. Leutenegger, S. and Edgington, J., A games first approach to teaching introductory programming. *Proc. 38th SIGCSE Technical Symp. on Computer Science Educ.*, New York, NY, USA, 115-118 (2007).

6. Moor, S., Music in MATLAB: a series of programming challenges for an introductory course. *Computer Applications in Engng. Educ.*, 18, **1**, 67-76 (2010).
7. Djordjevic, M., Java projects motivated by student interests. *Proc. 13th Conf. on Innovation and Technol. in Computer Science Educ.*, New York, NY, USA, 321-321 (2008).
8. Mahnič, V. and Gams, M., Some experiences in teaching introductory programming at the faculty level. *World Transactions on Engng. and Technol. Educ.*, 2, **3**, 441-444 (2003).
9. Richards, B., Experiences incorporating Java into the introductory sequence. *J. of Computing in Small Colleges*, 19, **2**, 247-253 (2003).
10. Benander, A., Benander, B. and Sang, J., Factors related to the difficulty of learning to program in Java - an empirical study of non-novice programmers. *Infor. and Software Technol.*, 46, **2**, 99-107 (2004).
11. Anik, Z. and Baykoç, O., Comparison of the most popular object-oriented software languages and criterions for introductory programming courses with analytic network process: a pilot study. *Computer Applications in Engng. Educ.*, 19, **1**, 89-96 (2011).
12. Knuth, D.E. and Moore, R.W., An analysis of alpha-beta pruning. *Artificial Intelligence*, 6, **4**, 293-326 (1975).