11/2 (2013), 165–180
tmcs@math.klte.hu
http://tmcs.math.klte.hu

Teaching Mathematics and Computer Science

Motivating Students with Projects Encompassing the Whole Duration of Their Studies

János Pánovics

Abstract. Based on my ten years of teaching experience at the University of Debrecen, I can say that students majoring Software Information Technology BSc have to face a number of difficulties during their studies. I think these difficulties root from two main problems: students are unmotivated and cannot sense the coherence between the knowledge acquired in the various courses. This paper tries to give some alleviation to both of these problems by the idea of introducing some long-term projects to students, which they can work on throughout their studies, dealing with a particular aspect of the projects in each course.

Key words and phrases: teaching methodology, motivation, coherence, long-term projects.

ZDM Subject Classification: D70.

1. Introduction

For the last few years, most of the students majoring in some area of computer science at the University of Debrecen have been having a hard time fulfilling the requirements of most nonbasic courses. This is partly because of the big number of students. (It is out of the scope of this paper why we need to have so many students.) First, we have to launch many practical courses for the same subject with many students in each of them. Due to this, we need many instructors (including student instructors), who have to deal with a lot of students and have much less time to deal with each of them individually. Second, a lot of the students come to our university not because of their interest in computer science but for other reasons (like parental pressure, the popularity of information technology, good job prospects, or simply because they misunderstand the program objectives), and therefore they are often undermotivated.

On the other hand, mass education is not the only reason for "mass failure" and poor performance. I believe that we, the instructors, do have some influence on the outcome of the education. The key is to find a way to pique students' interest. We can do this by assigning them tasks in which they are interested. Creating a two-player game with competitive artificial intelligence and a graphical front-end, writing a library information system that keeps track of data about books, patrons, and loans, or creating a web-based network analysis tool which computes different statistical data about network traffic may be such tasks. For example, we can read about the idea of using Reversi as a teaching tool in [9], teaching fundamental programming concepts via two-dimensional game development in [6], or using physical and virtual models of discrete games to help students learn the fundamental concepts and problem solving strategies in computer science in [1]. In particular, two computer games are used to teach the concepts of boolean expressions and recursion in [4]. Even abstract knowledge of mathematical logic can be presented through playful tasks [8]. Whatever the task is about, the secret is that it should be a large-scale project, which covers more (or even most) of the subjects students encounter during their studies. Throughout the project work, they apply the knowledge discussed in the lectures and practical courses of the related core subjects. They learn the applicability and usability of the topics of each subject as well as the problems emerging during the application of that knowledge. I think that if we can find appropriate assignments, we may achieve better performance not only in solving the assignments but also in the final examinations of the courses.

Defining assignments related to developing real-world applications has the following benefits:

- Compelling examples increase students' motivation.
- Via a complex project, students can practice a number of aspects of computer science.
- Using the same complex project in more courses will help students better understand the relationship between the knowledge behind those courses.
- Projects make computer science education more practice-oriented.
- Projects validate the theoretical knowledge acquired during the lectures and answer the question of how to use that knowledge.

I would like to emphasize that the idea of project-oriented education is already applied in most graduate (master) programs of computer science at most Hungarian universities. According to the current act on higher education, even undergraduate programs must contain some amount of project work. We can see a good example of this at the Eötvös Loránd University, Faculty of Informatics, where students can participate in a cooperative training for 16 credits [2]. The goal of the cooperative training is to provide students with the possibility of getting acquainted with the practical side of computer science under the supervision of experienced professionals at real companies in the software industry. Another example is the subject titled *Project Laboratory* at the Budapest University of Technology and Economics, Faculty of Electrical Engineering and Informatics [7] or at the University of Debrecen, Faculty of Informatics, where students may deepen their knowledge and get some experience in a specific field of computer science.

There are, however, some important differences between the proposed approach of "teaching with projects" and the above-mentioned examples:

- Both cooperative training and the *Project Laboratory* courses are independent from the core subjects of an undergraduate program in the sense that they are separate educational units. According to my proposal, the projects would form a part of the course materials of most core subjects, so students can work on projects in the frame of the existing subjects, and no separate subjects or trainings are required.
- While cooperative training and the *Project Laboratory* courses focus on only one or two areas of computer science, the proposed approach covers the topics of most core subjects.
- Cooperative training and the *Project Laboratory* subject both have strict prerequisites, i.e., they are based on knowledge acquired during earlier studies. However, using the proposed approach, students start working on projects as early as the first semester. This also means that instructors have to deal with a lot more students, who have not participated in project works before. On the other hand, instructors may also be inexperienced in project management, and they need to cooperate with one another in order to achieve a better result.
- Although cooperative training is a part of education, the institute forfeits its right to control the flow of the training and the assessments. Another drawback is that it is not so easy to find the necessary number of companies with appropriate projects outside the capital.

2. Current Program Requirements

Let’s first have a look at the requirements of the Software Information Technology BSc major. The program lasts for 6 semesters, and students have to gather a total of 180 credits according to the following list:

- 120 credits from core subjects
- 29 credits from compulsory elective subjects
- 11 credits from elective nonvocational subjects
- 20 credits from the thesis

Table 1 contains the full list of the core subjects with credit numbers, contact hours, prerequisites, and recommended semesters [3].

Subject code	**Subject name**	**Credit**	**Lectures**	**Seminars**	**Labs**	**Prerequisites**	**Recommended semester**
CS101	Discrete Mathematics 1	5	2	2			1
CS111	Calculus 1	5	2	2			1
CS401	Logic in Computer Science	5	2	2			1
CS201	Introduction to Informatics	5	2		2		1
CS202	HTML, XML	2			2		1
CS711	Computer Architectures	5	2		2		1
CS102	Discrete Mathematics 2	5	2	2		CS101	2
CS112	Calculus 2	5	2	2		CS111	2
CS131	Probability Theory and Statistics	5	2		2	CS101 CS111	2
CS421	Data Structures and Algorithms	5	2	2		CS201	2
CS301	Programming Languages 1	5	2		2	CS201	2
CS211	Operating Systems 1	5	2		2	CS201	2
CS411	Automata and Formal Languages	5	2	2		CS101	3
CS302	Programming Languages 2	5	2		2	CS301	3
CS212	Operating Systems 2	5	2		2	CS211	3
CS501	Database Systems	5	2		2	CS301	3
CS601	Introduction to Computer Graphics	5	2		2	CS101 CS301	3
CS141	Numerical Methods	5	2		2	CS102	3
CS441	Introduction to Artificial Intelligence	5	2	2		CS302 or (CS301 and CS401)	4
CS311	Programming Environments	2			2	CS302	4
CS321	Programming Technologies	5	2		2	CS302	4
CS721	Computer Network Architectures and Protocols	5	2		2	CS711 CS212	4

Subject code	Subject name	Credit	Lectures	Seminars	Labs	Prerequisites	Recommended semester
CS511	Database Administration	3	2			CS501	5
CS521	Technology of System Development	5	2		2	CS321	5
CS001	Thesis 1	10				CS321	5
CS451	Algorithm Design and Analysis	5	2	2		CS401 CS411	6
CS231	Internet Tools and Services	3	2			CS521	6
CS002	Thesis 2	10				CS321	6

Table 1: Core subjects

The compulsory elective subjects are divided into five subject groups (called blocks), each containing four subjects with a total of 16–18 credits. These blocks cover the following areas: Artificial Intelligence, Database Systems, Operating Systems and Networks, Computer Graphics, and Information Theory and Applied Mathematics. Students have to complete at least one subject from each of these blocks. The remaining credits needed for the 29 credits can be earned by completing other subjects from the blocks or additional elective vocational subjects launched by the faculty at the beginning of each semester.

As you can see, students have to take and complete at least 37 subjects during their studies, which is a rather big number for only 6 semesters. I think that some of these subjects should be a part of graduate programs, and others (the basic subjects) should get more emphasis with more contact hours.

However, even if we insist on this study plan, we may still find projects that involve a number of the listed areas. Let's now have a look at a couple of examples.

3. Some Possible Projects

If we take a closer look at Table 1, we can soon realize that even a medium-sized software development project needs some knowledge from at least three core subjects. We can state this based merely on the names of the subjects, without knowing the detailed topics of them and without knowing the goal of the application to be developed. The three most basic subjects, which are involved in every software development project, are *Introduction to Informatics*, *Data Structures and Algorithms*, and *Programming Languages 1*. However, most of the other core subjects may also be involved in a long-term project.

In this section, I would like to present two projects of medium difficulty, parts of which may be used as assignments from as early as the first semester.

3.1. Project #1: Creating a Reversi Application

Of course, Reversi may be replaced by any (not too difficult) two-player game here. The main goals of this project are the following:

- To learn some basic programming idioms in at least one programming language.
- To learn how to represent the data structures used during the implementation as well as the algorithms that work with them.
- To learn some basic artificial intelligence methods that are good enough to beat at least a weak human player.

For the assignments to make sense, I briefly introduce the rules of the game [5]. Reversi (or Othello) is a strategy board game for two players, played on an 8×8 uncheckered board with 64 pieces colored differently on each side, which correspond to the opponents of a game. The color facing up indicates which of the two players controls the square occupied by the piece. The game begins with the central four squares occupied: each player controls one of the diagonals. Players take turns placing pieces on the board with their assigned color facing up until neither can make a move (usually when all 64 squares are occupied). The player who controls the most squares is the winner. A legal move must be to an empty square and must bracket at least one of the opponent's pieces in a straight line between an existing piece of the player in turn and the newly played piece. Upon moving to that square, all of the bracketed opponent's pieces are flipped, in all 8 directions. If a player has no legal move, they must pass, and their opponent will move. If a player has a legal move, they must make it even if it hurts their game.

Here is the list of subjects that may be affected by this project, along with topics of interest and example assignments regarding each subject:

- *Discrete Mathematics 1*: This is a subject with topics from set algebra, linear algebra, number theory, and combinatorics. We can say that almost all projects require some mathematical knowledge, though not necessarily in-depth knowledge. In the seminars, students can be asked combinatorial questions regarding the Reversi game. Example assignments for this subject include the following:
  - *How many arrangements of an* 8×8 *board are possible?*

  – *How many games in a* 6×6 *board are possible?*

- *Introduction to Informatics*: During the lectures, students learn the basic concepts that are essential for everyone with a degree in computer science. They learn, among others, about hardware and software, data representation, and basic searching and sorting algorithms. In the laboratories, they practice data representation and start writing simple programs in C. As for the project, the instructors may show how integers, real numbers, characters, strings, or other basic data may be represented in the memory. Although this knowledge is not essential for creating our application, I agree with those who say that data representation is a basic building stone of informatics without which the operation of a computer cannot be understood. Another significant result of this subject is that students write their first C programs so they can begin experimenting with the language. I think it is very important to start writing programs as soon as possible because it takes some time to get accustomed to using the language features for someone who has never seen a high-level program code before. Example assignments for this subject include the following:
  – *Suppose we later want to write a function* $\mathsf{H(b,p)} = \mathsf{P(b,p)} + 8\mathsf{E(b,p)} + 64\mathsf{C(b,p)}$ *where* b *is the board,* p *is one of the players,* $\mathsf{P(b,p)}$ *computes the number of pieces* p *has on board* b*,* $\mathsf{E(b,p)}$ *computes the number of edge pieces* p *has on board* b*, and* $\mathsf{C(b,p)}$ *computes the number of corner pieces* p *has on board* b*. Suggest a representation of the value of* $\mathsf{H(b,p)}$ *considering its minimum and maximum possible value.*
  – *Create a well-formed text document containing the rules of Reversi as well as the schedule of a Reversi tournament.*
- *HTML, XML*: As the name suggests, this subject deals with the syntax and use of these two important markup languages. Students learn how HTML can be used to create simple, static web pages, and how XML can be used to store almost any kind of hierarchically organized data. The knowledge provided by this subject can be useful for any project because all projects may make use of a simple web page or an XML database. In our case, we can store, for example, a game flow in an XML file. Example assignments for this subject include the following:
  – *Create a simple static HTML website containing information about our future Reversi game (e.g., the game rules).*
  – *Design an XML data structure (DTD) for storing the game flow.*

- *Logic in Computer Science*: Mathematical logic is used in a number of areas in computer science. In this introductory course, students learn about first-order predicate calculus. It is a big problem that a lot of students do not see at this point why this subject is important, and where they can use the acquired knowledge in the future. The instructors should explain them that all programming languages use conditions, and that conditions are actually logical formulae with all of their properties. Students should know how logical operators (such as implication or the universal quantifier) can be implemented in a programming language that does not implicitly contain those operators. As no laboratory belongs to this subject, these tasks are usually completed only during *Introduction to Artificial Intelligence* lessons. Other concepts that also occur during programming are those of free and bound variables, which may be implemented using parameters and local variables, respectively. Mathematical logic also plays a role in other subjects like *Database Systems* or *Introduction to Artificial Intelligence*. Example assignments for this subject include the following:
  - *Create a new first-order language with syntax and semantics which can be used to express different elements of the Reversi game. The language may include functions like the number of each player's pieces or the number of empty squares, and atomic formulae, e.g., for deciding whether a particular square of the board contains a particular player's piece, the game is over, the player in turn has won, the player in turn can win in the next move, or one of the players has more pieces than the other.*
  - *Formalize some interesting assertions about the game as compound formulae such as "if there are no empty squares left, the game is over" or "all nonempty squares contain a piece of either Player 1 or Player 2."*
- *Data Structures and Algorithms*: The goal of this subject is to present the most popular abstract data structures (including files) and their different implementations to the students. In the seminars, students first learn about three searching and at least five sorting algorithms in detail with C implementations, and then practice the use of the most important data structures that have been talked about during the lectures. This is the first subject in time that has a greater direct influence on our project. From the application's point of view, one of the most important data structures is the array or its two-dimensional version, the matrix. It is because the board of the Reversi game and the pieces of the players can most conveniently be represented using an 8×8 matrix. It is interesting to mention how to implement the matrix

in a language (like C) that does not support multidimensional arrays. Of course, matrix is not the only data structure used in this project. We will later need, among others, a record to store the states of the game, which can be implemented as a class in an object-oriented language, or a nonbinary tree to represent a part of the game tree, which again can be implemented as a class. Example assignments for this subject include the following:

- *Implement an 8×8 matrix representing the board in C language. Write functions that execute simple operations on the board like setting all squares in a given row between two given columns to a given piece.*
- *Implement a stack for storing the board states after successive moves for undoing/redoing the moves.*

• *Programming Languages 1*: From a programming aspect, this is the most important subject, which has the most influence on any software development project. The lectures teach students all the concepts related to high-level programming languages, while in the laboratories, students should acquire the use of a specific procedural language. At our university, this language has been C for ages now because of its significance and because it serves as a base for other, object-oriented languages like Java. Learning a programming language via small sample programs is a good method for the beginning, but they are not enough for learning how to *use* that language. This is where a larger-scale application comes in handy. It not only inspires students to spend some time with programming but also makes them meet situations that otherwise would not come to the front. So in the laboratories, after learning the language itself (which should not take more than 4 weeks), students can create the first version of the Reversi application with the help of the instructor. Of course, students have to use the data structures learned in the parallel course *Data Structures and Algorithms.* After finishing the second semester, they may be entitled to say that they are able to create simple (but usable) applications in C. Example assignments for this subject include the following:

  - *Write a function in C that takes a board and a player as parameters and returns the number of pieces the given player has on the given board.*
  - *Write a procedure that takes a board and a player as parameters, reads the given player's next move from the keyboard in a loop until the user enters a legal move, and updates the board according to the move. The code that checks the legality of a move should be placed in a distinct function.*

After successfully completing a number of such assignments, students will have their first working version of the Reversi game, which is able to store the state of the game, draw the board to a character-based console, read the players' input from the keyboard, check the legality of the moves, check if the game is over, and print the result. As an optional assignment, they may improve the first version of the game with the ability to play against the computer. The computer may choose its move randomly in this version. The user should have the possibility to decide whether they want to play with the computer and if so, select the starting player.

- *Programming Languages 2*: The aim of this subject is to get students know the ins and outs of the object-oriented (OO) paradigm. The lectures also touch functional programming, but in the laboratories, they only have to learn one or two object-oriented languages (currently Java and C#). As newer and newer concepts are introduced in the lectures (classes, inheritance, polymorphism, interfaces, etc.), students can gradually rewrite the code of our game application in Java or C#. This way, they can compare the procedural and OO version of the same program and much better see the benefits of the OO paradigm. Example assignments for this subject include the following:
  - *Recode the first version of the game in Java and/or C#.*
  - *Try to rework the result so that it uses more and more OO programming idioms, classes, inheritance, interfaces, and OO data types (especially for collections).*
- *Introduction to Artificial Intelligence*: The lectures of this subject are about state-space representation, various search algorithms, problem reduction representation, and look-ahead algorithms for finding the best move in a two-player game. In the seminars, students first create state-space representations for various problems. After this, they learn how to implement logical formulae in Java or C#, then create a class hierarchy for the most popular search algorithms, and finally implement the minimax and negamax algorithms for two-player games. Although theoretically the subject has no laboratory courses, students still use computers and write programs during the seminars in the second part of the semester. Needless to say, this subject is of great importance concerning our project. By the end of the semester, students are able to build the minimax algorithm into the application so that human players can play against the computer. The instructors may even organize a contest among the students' programs to further motivate them to write the best

possible heuristic function. Example assignments for this subject include the following:

  – *Create the Java or C# code implementing all the logical operators of first-order logic.*
  – *Augment the latest version of your program by integrating a minimax (or negamax) algorithm and eventually, alpha-beta pruning.*

- *Programming Environments*: This is a laboratory-only subject, which focuses on the usage of integrated development environments, debugging, CASE tools, the control of compilation, and using libraries. The instructors can show students how to detect different semantic errors in the Reversi application with the help of the debugger of Netbeans or Visual Studio. Students can also learn how to use a CASE tool, for example, to create the Java code from a UML class diagram and thus shorten the coding time. As you can see, there are quite a few possibilities to experiment with our project throughout this course. Example assignments for this subject include the following:
  – *Create a statically/dynamically linked library from the AI part of the Reversi application so that it can be used later with other applications as well.*
- *Programming Technologies*: The lectures of this subject deal with different programming methodologies, reuse-oriented development, the role of abstraction, programming idioms, design patterns, good programming styles, refactoring, testing, validation and verification, software metrics, and software quality assurance. In the laboratories, students learn about UML, advanced exception handling, using C# delegates, multithreading, reflection, working with metadata (annotations in Java or attributes in C#), using the Java API or the .NET framework, creating graphical user interfaces (e.g., using Swing), JavaBeans, database connectivity from Java and C#, network handling, processing XML files, internationalization (i18n), and using regular expressions. It can be seen from this enumeration that this subject covers a very wide area of software development. Because of this, instructors can show students a lot of exciting aspects of programming. Example assignments for this subject include the following:
  – *Create a GUI for your application using some visual form designer.*
  – *Make the program multilingual using i18n.*
  – *Extend the application with the capability of loading games from and saving games to XML files or a database (e.g., with JDBC).*

- *Computer Network Architectures and Protocols*: The lectures of this subject cover the theory of networking based on the ISO OSI model and the most popular protocols of each layer. Laboratories are used, among others, to create and implement new application layer protocols. Example assignments for this subject include the following:
  - *Create a client/server version of your application. This means that the server side runs on some host, and clients connect to it through TCP/IP and a new application layer protocol, which controls the game flow. As a bonus, you may write the server side using multithreading so that each client connection starts a new thread, which is responsible for the communication with that client.*
- *Technology of System Development*: The lectures are about the process of software development, process models, functional and nonfunctional requirements, system models, requirement analysis, design, standards (UML, MDA), service-oriented architecture, and agile software development. In the laboratories, students create different UML diagrams and ISO documents. They also learn how to use a version control system and developing in teamwork. Example assignments for this subject include the following:
  - *Create different kinds of UML diagrams (e.g., a class diagram, use case diagrams, or state diagrams) concerning our project.*

3.2. Project #2: Creating a Library Information System

This project differs from the Reversi project in that it does not require artificial intelligence but requires much deeper database knowledge. Of course, we can again replace the word "library" by any other institution (e.g., a hospital, a shop, a school, etc.); the point is that we need to keep track of a big amount of data and be able to execute (possibly complicated) queries via a user-friendly web-based interface or a thick client program.

Most of the depicted connections between each subject and the Reversi project apply to this project too. The differences are the following:

- *Data Structures and Algorithms*: For this project, instructors may show students how to create abstract record data structures for storing the books' data, the patrons' data, etc., and how to implement them using the *struct* type in C. The other thing students may learn is the different abstract file formats (serial, sequential, direct, indexed, etc.) with which these records can be stored.

- *Programming Languages 1*: The first version of the application can work with files instead of databases so we can end up with a C program that can read and write data from and to text files or binary files. This way, students will see the big difference between file management and database management during the *Database Systems* course.
- *Programming Languages 2*: This project probably needs a little more complicated class hierarchy than the Reversi project so students can better practice inheritance, polymorphism, or interfaces. On the other hand, they can also realize that file management is somewhat more convenient in Java or C# than in C.
- *Database Systems*: The Reversi application did not use databases (unless we added the capability of loading and saving games). However, database management is a crucial building stone in the Library project. In the lectures of this subject, students learn about the basic concepts of database systems as well as the relational, ER, EER, and ODMG data models, with special emphasis on the relational model. In the laboratories, students use Oracle SQL to acquire the usage of SQL DML, DDL, and DCL. This course is very important for our project. Students have to practice complex SELECT statements in our Library database to be able to build arbitrary queries into our application.
- *Database Administration*: This is a lecture-only subject about the role of the database administrator, creating a database environment, handling metadata, storage management, distributed databases, database security, archiving and recovery, preparing for catastrophes, database performance, and change management. Although this subject lacks laboratories, the lecturer can show examples of the above topics concerning our Library database. Examples from a well-known system always helps better understand the underlying knowledge than examples from different, independent, unknown systems (or even from one single but unknown system).

3.3. Further Subjects and Projects

Subjects in Table 1 not mentioned so far are less important from a programmer's point of view, or at least the knowledge behind them is less used in real-world applications. There are two more subjects with laboratories during which students write programs. One is *Introduction to Computer Graphics* where they create programs, among others, for drawing simple graphic shapes like lines or circles, or for drawing three-dimensional shapes using parallel or central projection. Students can make use of the knowledge acquired during this subject in projects like creating a computer game with advanced graphics. If a student wants to work in this area, then this subject is an essential base for them, which must be followed by other, advanced subjects dealing with computer graphics like those in the Computer Graphics block.

The other core subject where students have to write programs is *Numerical Methods*. In this subject, they learn about function approximation, numerical differentiation, numerical quadrature, various methods for solving linear and nonlinear equations and equation systems, matrix factorization and inversion, computing determinants, and approximation of the eigenvectors and eigenvalues of matrices. They also learn to use software like MATLAB or the LINDO API. Some of the methods mentioned in the lectures are coded in the laboratories, while others are homework assignments. The knowledge provided by this subject along with other mathematical subjects like *Discrete Mathematics 1/2*, *Calculus 1/2*, or *Probability Theory and Statistics* can be applied in projects with some mathematical background. As an example, an application for various kinds of statistical analyses may be such a project. To further narrow it, someone may want to write a program that provides different statistical data from the electronic administration system used by the institution. This example also has to do with data mining or even data warehouses, which are areas covered in one of our graduate programs.

There is some sort of programming in the laboratories of *Computer Architectures* too. This subject overlaps with *Introduction to Informatics* because both deal with data representation, but *Computer Architectures* is more about the abstract architecture and operation of a computer. To help students better understand how computers work at lower levels, they learn some assembly programming during the laboratories. Students majoring Engineering Information Technology could make more use of this knowledge, although, interestingly enough, they do not have a laboratory course for this subject. Nevertheless, assembly programming comes in handy in projects requiring low-level programming such as writing drivers for different hardware components.

Subjects like *Automata and Formal Languages*, *Algorithm Design and Analysis*, and *Internet Tools and Services* have rather theoretical significance from a programming aspect. For example, if someone wants to write a compiler or just a parser for some language, then they can use the knowledge acquired during the courses of *Automata and Formal Languages*. However, automata can also be used in everyday programming, e.g., when coding an event loop using state machines.

Last but not least, subjects *Operating Systems 1/2* are about the architecture and functions of operating systems. In the laboratories, students learn to use and a little to administer Windows and Linux, today's two most popular operating systems. These are more practical subjects, but they have only little to do with programming. However, students learn during these subjects how to write scripts using batch files in Windows or shell scripts in Linux. They can also benefit from this knowledge when programming in other script languages like JavaScript.

4. Conclusion

I believe that learning all the aforementioned knowledge can be much more entertaining for the students by developing one or two larger-scale applications throughout their studies (even if in teamwork) than just writing small sample programs for every different area of software development. With one complex project or with two or three medium-sized applications, we can cover nearly every aspect of the development process and give students a comprehensive example of software engineering. If our faculty introduced this "learning via projects" approach of teaching, students would be a little more motivated and would more likely see the coherence between the topics of the wide range of subjects. On the other hand, this approach requires some extra work on the instructors' part: they need to find appropriate real-world applications that could become the projects, cooperate with one another on distributing the different parts of the projects among the various courses, and a lead instructor should be designated as the person in charge of these tasks, who has an oversight on all subjects in the study plan.

Of course, we cannot expect a radical improvement in students' performance just because of such a minor change in our teaching methodology. Decreasing the number of students or redesigning the program's study plan would have a much bigger effect on it. Although the faculty has little or no influence on the number of enrolled students, we could still initiate the supervision of the program requirements of the Software Information Technology BSc major.

Acknowledgments

The publication is supported by the TÁMOP-4.2.2/B-10/1-2010-0024 project. The project is co-financed by the European Union and the European Social Fund.

I would like to thank Magda Várterész for all her useful advice.

References

[1] J. Albornoz Bueno and R. A. Chaparro Aguilar, The learning of fundamental concepts and problem solving strategies in computer science, through the experimentation and classroom research with discrete games, *Proceedings of the 9th International Conference on Engineering Education*, San Juan, Puerto Rico (July 23–28, 2006).

[2] Cooperative training at the Eötvös Loránd University, Faculty of Informatics, `http://www.inf.elte.hu/karunkrol/oktatas/kepzeseink/kooperativkepzes/Lapok/altalanosleiras.aspx`.

[3] Degree requirements for the Software Information Technology BSc major at the University of Debrecen, `http://www.inf.unideb.hu/oktatas/?cat=&site=hallgato/nappali/oklevel_kovetelmeny/pti_2007`.

[4] Jeffrey Michael Edgington, *Toward using games to teach fundamental computer science concepts* (doctoral dissertation), University of Denver, 2010.

[5] István Fekete, Tibor Gregorics, and Sára Nagy, *Bevezetés a mesterséges intelligenciába*, ELTE Eötvös Kiadó, Budapest, 2006.

[6] S. T. Leutenegger and J. M. Edgington, A games first approach to teaching introductory programming, *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education* **39**, no. 1, ACM Press (2007), 115–118.

[7] Project Laboratory at the Budapest University of Technology and Economics, `https://www.vik.bme.hu/kepzes/targyak/VIAUA354`.

[8] Raymond M. Smullyan, *Gödel's incompleteness theorems*, Oxford University Press, New York, 1992.

[9] D. W. Valentine, Playing around in the CS curriculum: Reversi as a teaching tool, *Journal of Computing Sciences in Colleges* **20**, no. 5 (2005), 214–222.

JÁNOS PÁNOVICS
FACULTY OF INFORMATICS
UNIVERSITY OF DEBRECEN
H-4028 DEBRECEN, KASSAI ÚT 26.
HUNGARY

E-mail: `panovics.janos@inf.unideb.hu`

(Received October, 2012)