Egyetemi doktori (Ph.D.) értekezés tézisei

The Appearance of Multiparadigm Programming Languages in the Teaching of Artificial Intelligence

A multiparadigmás programozási nyelvek megjelenése a mesterséges intelligencia oktatásában

Pánovics János

Témavezető: Dr. Fazekas Gábor

Debreceni Egyetem
Természettudományi Doktori Tanács
Matematika- és Számítástudományok Doktori Iskola

Debrecen, 2013

Contents

CHAPTER 1

Introduction and Motivation

The main goal of my dissertation was to come up with a new methodology that helps make teaching programming-oriented computer science in higher education more effective. The new methodology is based on long-term projects spanning the whole duration of students' studies, forcing them to apply the knowledge acquired during the project-related subjects. The presented approach may have numerous advantages over the currently used teaching methodology. I also give a couple of sample projects with example assignments in subjects of the Software Information Technology BSc major at the University of Debrecen that may be related to those projects.

As a small part of the proposed methodology, I put special emphasis on the teaching of the subject *Introduction to Artificial Intelligence*. The success of this subject is based on the "state space" approach with precise mathematical background, which is applied in the education of AI at the University of Debrecen. I was influenced by this approach when creating the sample programs. The topics of the subject include the state-space representation of single-agent problems and two-player games as well as numerous algorithms searching for a solution of a problem or finding the best move in a given state of a two-player game [5, 6, 14, 11]. I propose the introduction of multiparadigm languages on the seminars of subjects dealing with artificial intelligence in practice. Giving more implementations of search algorithms using different approaches may help students understand the logic behind those algorithms. I give detailed examples of such implementations in C# and F# and also compare those implementations.

Programming contests can be a fairly motivating factor for students for learning to use the features of a programming language, the basic and advanced data structures, or the basic and advanced algorithms processing them. Together with two of my colleagues, we have developed a web application which is able to manage programming contests as well as to evaluate solutions to programming assignments submitted by students. I

briefly introduce this application and describe its benefits against other similar software and our experience with it.

To summarize the above, my thesis sets the following goals:

- to design some sample long-term projects to be assigned to students in the frame of the new methodology,
- to create different sample implementations of the most popular AI search algorithms in C# and F#, including those that compute the next move in a two-player game,
- to compare the different implementations of these algorithms (from purely object-oriented to mostly functional),
- to give some examples of how to create C# and F# implementations of specific problems and games that can be fitted into the above-mentioned algorithms,
- to argue for using multiparadigm programming languages in the teaching of artificial intelligence based on the above implementations,
- to share our experience with the programming contests organized by the Faculty of Informatics at the University of Debrecen as well as the applications managing them.

CHAPTER 2

New Results

2.1 A New Methodology for Teaching Computer Science

One of the main areas of my research was trying to find an answer to the following question: "Why do most of our students majoring Software Information Technology BSc have a hard time fulfilling the requirements of most nonbasic courses?" Based on my ten years of teaching experience at the University of Debrecen, I can say that our students have to face a number of difficulties during their studies. This is partly because of the big number of 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.

However, 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 efficiency 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 [18], teaching fundamental programming concepts via two-dimensional game development in [10], 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 [17]. In fact, nowadays one can hardly find a renowned journal or conference on computer science education without a publication about teaching via real-world projects. As examples, we can mention the journals *Computers & Education* (impact factor: 2.621, publisher: Elsevier) or the *Journal of Computer Assisted Learning* (impact factor: 1.464, publisher: Wiley), and conferences like the *3rd Annual International Conference on Computer Science Education: Innovation and Technology* (held on November 19–20, 2012, in Singapore) or the *Consortium for Computing Sciences in Colleges — Northeastern Region* (held on April 12–13, 2013, in Albany, New York).

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 a better performance not only in solving the assignments but also in the final examinations of the courses.

If we have a look at the list of the core and compulsory elective subjects of the Software Information Technology BSc major at the University of Debrecen [3], we can soon realize that even a medium-sized software development project needs some knowledge from at least three core subjects: *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. We can easily find exciting assignments similar to some existing real-world applications that may play the role of such a long-term project. Once we cooked up the goals of the project, we just have to find some assignments used in each of the related subjects. In my opinion, if students work on a complex project with the development process encompassing almost the whole duration of their studies, they will become more motivated and better see the relationship between the knowledge learned in each course.

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 [13] 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. Project-based education is also applied in foreign institutions; for instance, at the University of Paderborn, teams of three have to develop operating systems during one of the undergraduate courses.

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.

  Thesis 1. I worked out a novel methodology for teaching computer science in undergraduate programs. The new methodology involves using long-term software development projects built in the practical courses of most core subjects, resulting in exciting real-world applications. Although a little extra work is required from the instructors' part, students would be more motivated and have a better oversight on the coherence between the knowledge acquired in he various courses. I also presented some sample projects along with example assignments for the practical courses of each related core subject.

2.2 Using Multiparadigm Programming Languages in the Teaching of Artificial Intelligence

Until recently, programming was about using pure, single-paradigm techniques. However, nowadays programming languages tend to converge to one another, i.e., features of one paradigm are appearing in languages based on another paradigm [15, 16]. This process can be observed primarily in the area of imperative, object-oriented, and functional programming languages. For example, C# or D are based on the imperative C language, extended with object-oriented and functional language elements. Another example

is CLOS, which is a functional language based on Lisp, with added object-oriented features. Such languages are called *multiparadigm programming languages*.

The advantage of these languages is that the programmer may select the paradigm best suited for solving a particular problem. We may even choose to create different parts of the same program using techniques from different programming paradigms. If a small part of a large application requires high efficiency, imperative code is used, reusable types are created in an object-oriented way, while in some domains of computation, we choose the functional approach because solutions to problems in these domains can be expressed in a more succinct way compared to using imperative code.

I believe that multiparadigm languages can be a great help in the education of computer science. Instructors may show different versions of the same algorithm depending on the paradigm(s) used to implement it, and later students may decide which paradigm(s) to choose in a homework assignment, an exam, or at work, depending on the problem specification and the student's programming experience. The project-oriented teaching methodology is perfectly suitable to illustrate the advantages of each paradigm. Students may solve the same assignment using more paradigms, compare the features and limitations of the different paradigms based on the resulting pieces of source code, and then select the most advantageous parts from each of them. Of course, we don't necessarily need to use multiparadigm languages for this purpose, we can also use multiple single-paradigm languages, but this way, we don't have to teach and students don't have to learn yet another language just for coding a particular algorithm.

During my research, I examined the possibility of using F# as a new multiparadigm programming language for coding different algorithms in the area of artificial intelligence (AI). I chose this area for three main reasons. First, I used to be an instructor of the seminars of the *Introduction to Artificial Intelligence* course at the University of Debrecen [19]. Second, AI and search algorithms, in particular, seemed to be a field of computation where we can make use of functional programming because some substantial parts of these algorithms (like, for example, checking operator preconditions) are essentially functional. And third, I had a couple of imperative and object-oriented implementations of the algorithms in question, which proved to be a good starting point for creating the F# versions [8, 7].

It is not hard to prove that functional programming has a great ad-

vantage in solving AI problems compared to imperative programming. As I showed in my paper [12], even the simple n-queens puzzle has a much more concise solution in F# (using only purely functional programming elements) than in C or C#. However, when it comes to efficiency and reusability, purely functional code may not be the best solution to some problems.

Besides an object-oriented C# implementation, I have also created three versions in F# in case of search algorithms for single-agent problems and two versions in case of search algorithms on game trees. My goal with creating these implementations was to give students more approaches to understand the same pseudocode. Although I do not know if they can better acquire the operation of these algorithms by thinking functionally, giving more than just one implementation cannot be harmful. Here is a partly subjective comparison of the four implementations of search algorithms for single-agent problems from various aspects.

- *Code metrics*: Despite that we cannot use the same metrics in case of a functional language as in case of imperative languages, I will use three metrics here that may be relevant to both C# and F#: lines of code, number of classes, and number of functions (including methods). Table 2.1 summarizes the values of these metrics in the different implementations.

	C#	F# ver. 1	F# ver. 2	F# ver. 3
Lines of code	842	537	536	417
Number of classes	15	13	6	4
Number of functions	49	43	43	34

Table 2.1. Some code metrics.

The *lines of code* metric denotes the number of lines in all source files, including empty lines, but not including the source code of any specific problems. The *number of classes* includes all classes defined in the source code but does not include exception classes, enumerations, and `IComparer` classes. The *number of functions* includes all functions and nonabstract method implementations, including constructors, but does not include lambda expressions.

As you can see, the third F# implementation is half the size of the

C# implementation. Of course, this difference comes mainly from the compact syntax of the F# language. The other reason for the F# implementations being shorter is that they do not contain two classes from the C# code, which take 57 lines of code.

The decrease in the number of classes is a result of making the code more functional. The first F# implementation has two classes less than the C# version as I mentioned above. In the second version, the seven concrete algorithm classes are replaced with seven functions. In the third version, the two abstract search classes are also converted to functions. To make the code purely functional, we would have to get rid of the remaining classes too, but it would not result in a shorter or more readable code. For example, with the use of classes, it is easy to write *reusable* code for an operator application: we just have to call the `Apply` method of the abstract `State` class without knowing how it is implemented in the concrete class representing the states of a specific problem. And this is exactly what the constructors of the two `Node` classes do. Actually, the two `Node` classes might be replaced with record types because these classes are not inherited by any other classes, but again, it would not be more readable, and on top of that, we would not be able to use such .NET methods as `Contains`.

The number of functions are very similar in each implementation. It is because each method in a class maps to a function in a classless implementation; even constructors have corresponding functions. The third F# implementation, however, contains somewhat less functions than the other three. The reason for this is that it lacks the seven overrides of the `ToString` method in each of the concrete algorithm classes (or objects). There is only one `printSearchInfo` function instead. The same is true for the `Expand` and `Search` methods in the graph search algorithms. All of the corresponding functions of these methods contain `match` expressions based on the algorithm used for searching. This shows well the difference between the object-oriented and the functional approach to the problem of introducing a new subclass of a base class and introducing new functionality to the subclasses. The OO approach is better if we want to introduce a new subclass because we do not have to touch the existing subclasses, just write the new class with all the functionality inherited from the base class. In FP, we need to add a new branch to all of the `match` expressions. However, FP is better if we want to add new functionality to

the existing subclasses because we just have to write a new function with a similar `match` expression to the existing ones. In OO, we need to extend the base class with a new method and all of its subclasses with method overrides.

- *Mutable data structures used*: There is no difference in this aspect between the implementations. Although purely functional programs use no mutable data at all, I used some mutable data structures in each implementation. If we want, we can replace the type of any or all of these data collections with F#'s immutable `list` or `seq` data type. The resulting code would be of the same size, but it would be less efficient because the recursive functions in the `List` and `Seq` modules are slower than the corresponding .NET methods. It is particularly true when elements are added to or deleted from these collections: in case of an immutable data structure, we have to copy the original collection with a slight modification in its elements. This is the reason why I used .NET collections instead of F#'s immutable data types for storing these collections.

- *Functional language constructs used*: The C# implementation does not contain any functional constructs, it is purely object-oriented. The first F# implementation uses tail-recursive functions and sequence operations as a replacement for loops. Although object expressions are not functional language constructs, the second F# implementation is full of them as a substitute for subclasses. The third F# implementation contains the most functional elements: a discriminated union is used for storing the algorithm type, `match` expressions are used to deal with it, some functions are used as first-class values, and it also features a higher-order function.

- *Efficiency*: As functional languages are more abstract than object-oriented languages, they need a more complicated runtime environment. This is the main reason why functional programs are generally less efficient than object-oriented programs, even if they are compiled and not interpreted. I ran the presented main programs of all implementations on the same computer, a Gigabyte T1028X TouchNote netbook with Intel Atom N280 CPU at 1.33 GHz and 1 GB of RAM, and all programs compiled with Microsoft Visual Studio 2010: the C# program finished in less than half a second, while the F# pro-

grams all ran for about 7 seconds. As I wrote, it would have been even worse if immutable F# data types were used.

The observations made regarding the various implementations of search algorithms for single-agent problems are valid for search algorithms on game trees too, although, the difference is smaller in the latter case. The same is true for the implementations of specific problems or games.

As a final conclusion, my opinion is that it is not worth insisting on one or the other paradigm if we can use more of them within one program. Functional code is sometimes more abstract, more readable, or just shorter than its object-oriented counterpart. On the other hand, OO code is usually more efficient and sometimes more reusable than its functional counterpart. This is why I think multiparadigm languages like F# can be more advantageous mainly in large-scale applications but also in smaller programs. Students are different, so some of them may better understand the algorithms based on a more functional approach than on a pure object-oriented code. For example, the `Seq.fold` function using a lambda expression with an accumulator parameter may better describe how minimum or maximum selection is performed for students who think recursively (functionally).

Thesis 2. I created a possible course guide for the practical course of the subject *Introduction to Artificial Intelligence*. This guide contains some sample implementations of the most popular search algorithms as well as some specific problems and two-player games in C# and F# languages. I propose using multiparadigm programming languages in this course because there is no single paradigm equally appropriate for every problem, and because instructors (and later students) may choose the paradigm best suited for them to solve a particular part of a problem.

2.3 Managing Programming Contests with ProgCont

Programming contests can be a strong motivating factor. We have been organizing at least one in-house contest per semester for more than ten years now, and I can say that there are at least a couple of students whose interest is piqued by those contests. During these ten years, we developed two applications for evaluating the contestants' submissions on-line. The first one is called Programming Contest Result Manager (PCRM), and

it is an e-mail-based console application, while the second one is called ProgCont, which is a web application with a client/server architecture. PCRM is described in detail in [9, 7], now I introduce ProgCont along with our experience with it.

The role of educational contests for students is to lead them into a deeper acquaintance with a specific field of their studies. Contests give students personal objectives that stimulate them to work on their own. Apart from being stimulating, contests have a positive effect on students' educational results—participating in a remarkable contest or finishing in a good position a couple of times may contribute to their professional experience. Additionally, educational contests may help nurture professional relationships.

Students of the University of Debrecen have been participating in the Central European regional rounds of ACM International Collegiate Programming Contest since 1995 (although the university did not enter for the contest between 1998 and 2000). As a student in the second year of my studies, I was lucky enough to be a member of the team that advanced from the local round to the regional in 1995. Since 2001, I have been acting as an organizer and a member of the judge of local rounds of ACM ICPC as well as other programming contests.

In earlier times, organizing the local university rounds was encumbered by the fact that we had to check the solutions submitted by the contestants "by hand," which, beyond inconvenience, hindered the efficient work of the judge and involved a number of possibilities of making mistakes. To find a solution to these problems, we decided to create an application that can process a large amount of submissions both in real time and off-line. Together with my colleague Kósa Márk and an agile student, Gunda Lénárd, we made an e-mail-based console application called *Programming Contest Result Manager* (PCRM) in 2004, which helped us evaluating submissions not only from contestants during a contest but also from students submitting solutions to homework assignments of a particular course, such as *Introduction to Artificial Intelligence*.

We used PCRM for a couple of years, but later, it failed to comply with our newer and newer expectations. For example, we wanted to have a user-friendly graphical interface for the application, and would have preferred a web interface for both the contestants and the judge. Another drawback was that it used POP3 protocol for retrieving the submitted solutions, which made it difficult to manage and use. We decided not to rework our

existing application but to test some third-party programs (by that time, we could find a couple of programs on the Internet that seemed to meet our needs) and at the same time, get some talented students to develop a brand-new web application as a thesis work. As a result, an ASP.NET-based web application was born in 2009, created by a student, which was quite usable but contained some bugs, and after the student left our institution, no one could maintain it anymore. The best third-party program found and tested was *PC*^{2} (Programming Contest Control System) developed by the California State University, but it too lacked some needed features.

This is how *ProgCont* came to life in 2011, created by Kádek Tamás, Kósa Márk and myself. It is by far the most usable and most robust utility for supporting programming contests we have ever met. The application is continuously under development—last time we had to extend it to support a new kind of contest, the Regional Team Contest of the Faculty of Informatics, organized for the first time in November 2012 for high schools and colleges of five nearby counties, which used a different evaluation system from ACM-like contests.

The ProgCont system consists of four key components: problem catalog, contest database, controller web application, and solution evaluator clients (see Figure 2.1).

The *problem catalog* contains all resources related to each problem, such as the problem description (sometimes in multiple languages), the figures in it, further (possibly downloadable) content, and the test cases used to evaluate solutions. Test cases are stored in one or more files. The way of testing the correctness of solutions can be configured to one of the following options: the program can either check if the output produced by the submitted solution is completely equal to a pregenerated output file, or call an external tool that analyzes the output and returns whether it can be considered correct. Time limits associated with the execution of the submitted program can be set for each test case separately. The problem catalog can be formed using the directory and file structure of the operating system. Each problem has a distinct folder, in which an XML file contains the problem description and a ZIP file contains the test cases, the pregenerated output files, and the testing parameters. Additionally, all other files referred to in the problem description are in this folder.

The *contest database* describes the relationship between the contest, the problems, the solutions, and the solution evaluations. Each contest comprises some problems selected from the problem catalog, in case of

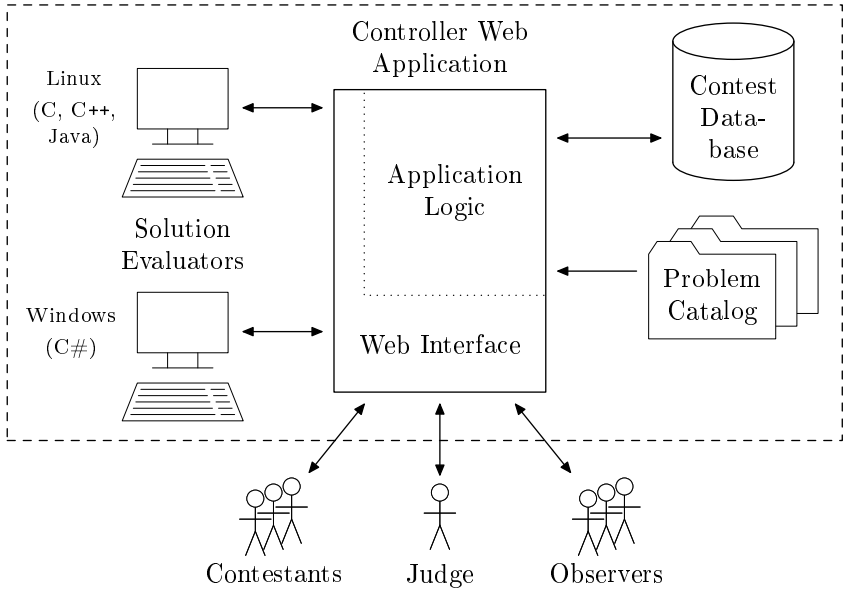


Figure 2.1. The architecture of ProgCont.

team contests, the members constituting the teams, and technical data regarding the flow of the contest, such as the start time and end time of the contest, or the allowed programming languages for each problem. The database stores the solutions submitted by the contestants and the result of each evaluation, which comes from one of the solution evaluator clients. All of this information is stored in a PostgreSQL database.

The flow of the contests is controlled by the *controller web application*. Contestants can browse the problems, submit the solutions, and learn about the results of evaluations and the current ranking using the web application. Observers (e.g., coaches of the contestants or guests) may also follow the current ranklist. The judge can set certain parameters of the contest via this web interface too. These parameters include the supported programming languages for each problem, the number of points each problem is worth, or the amount of penalty time used when computing the score of the contestants. In addition, the web application is responsible for scheduling the evaluation of the submitted solutions, i.e., distributing them among

the solution evaluator clients currently connected to the system. This distribution is based mainly on the information sent by the clients about the programming languages they support. For running the controller web application, Apache Tomcat web application server is used. Communication between the web application and the users (contestants and the judge) is secured by the SSL (HTTPS) protocol. The web application uses JDBC for accessing the contest database.

Solution evaluator clients are separate applications which periodically (every 5 seconds) check whether there is an available solution waiting for evaluation that they can handle. If so, they first try to compile the program code passed on to them by the controller web application using a preset compiler with preset options. If the compilation succeeds, the program is run for each test case using a preset runtime environment with preset options. The current test cases are downloaded from the problem catalog via the web application whenever the ZIP file containing them changes (and of course, the very first time they are used). During the execution of the program, the solution evaluator client takes into consideration the time limit set for the given test case. If the program stops within the time limit, its output is analyzed depending on the preset method of testing its correctness: it is either compared to the downloaded output file, or passed on to the external evaluator tool. Finally, the cumulative result is sent to the controller web application.

As different programming languages suit different operating systems, the solution evaluator clients were implemented as platform-independent Java applications. This way, they can (and should) be run on distinct (possibly virtual) computer(s) from that of the controller web application and the other evaluator clients, thus not endangering their operation, should a harmful code disrupt the runtime environment of a particular client. For example, a Windows-based client will compile and run C# code, while for C, C++, Java, or Pascal, a Linux-based client may be used. The more solution evaluator clients are used, the more evaluations can be performed at the same time.

We organized the first contest controlled by the ProgCont application on October 2, 2011. With respect to our original goal, it was the local round of ACM ICPC of that year. Making use of the ability to parameterize the contests, a short-term individual contest took place on February 6, 2012, among students of the course *Problems in Programming Contests*. In the same semester, we could help students deepen their knowledge in three

different courses using contests lasting for more than a month: *Problems in Programming Contests*, *Programming Languages 1*, and *Introduction to Artificial Intelligence*. After that, we organized an ACM-like contest on May 6, 2012 (which was the preliminary round of ECN International Programming Contest in Târgu Mureş, Romania), another local ACM contest on October 7, 2012, and last but not least, the Regional Team Contest of the Faculty of Informatics on November 25, 2012.

> **Thesis 3.** Programming contests appear to be a further motivating factor for students. Together with two of my colleagues, we have specified the conceptual principles of a contest management software and then designed and developed a web application, which is equally suitable to manage ACM-like and other types of contests and to evaluate the submitted solutions of homework assignments of different courses. *ProgCont* is a platform-independent, parameterizable contest management system, which well fits into the proposed project-oriented approach of teaching.

CHAPTER 3

Summary

To summarize my results, I hope I managed to create some aids and tools that can make students' life easier during their studies. One such aid may be to assign them long-term projects, which they can work on from as early as the first semester. A real-world application is always more exciting than small program snippets, so a long software development process may improve students' motivation for learning and programming. They can create the relevant parts of the application in each course related to the project under the instructors' supervision. Another beneficial effect of this approach is that students will better see the coherence between the knowledge acquired in the various courses. I presented two such long-term projects with some possible assignments in each related subject.

I created a course guide for *Introduction to Artificial Intelligence* or other AI-related courses. It is made up of various C# and F# implementations of search algorithms for single-agent problems and for two-player games. The difference between these implementations lies in the amount of functional programming constructs used. I also created some C# and F# implementations of some specific problems and games. Students and instructors may select the version best suited for them, depending on their way of thinking and their knowledge of programming. This way, students may better understand the operation of search algorithms presented in the lectures. As a result of creating and comparing these implementations, I drew the conclusion that the use of multiparadigm programming languages during AI courses may be beneficial both for instructors and for students.

Programming contests can also be a great motivating factor for students. We have been organizing ACM-like contests for more than ten years now, and I can say there are quite a few students who would never miss an opportunity to try a fall with others in a competition. Not just for their sake, but we needed an application for managing ACM-like and possibly other kinds of programming contests as well. As we could not find a contest

management software satisfying all our needs, two of my colleagues and I have decided to develop a brand-new web application for managing programming contests. *ProgCont* is designed to have a modular architecture and be parameterizable and easily extensible with new supported programming languages.

4. FEJEZET

Bevezetés és célkitűzések

Az értekezésem fő célja egy új módszertan kidolgozása volt a programozás-orientált informatika felsőfokú oktatásának eredményesebbé tétele érdekében. Az új módszertan a hallgatók teljes tanulmányi idejét felölelő hosszú távú projekteken alapul, amelyek a legtöbb kötelező tárgy ismereteit magukban foglalják. A bemutatott megközelítésnek számos előnye lehet a jelenleg használt oktatási módszertanhoz képest. Ismertetek néhány mintaprojektet is, valamint hozzájuk kötődően néhány mintafeladatot, amelyeket a Debreceni Egyetem programtervező informatikus BSc szakának a projekthez kapcsolódó kötelező tárgyai keretein belül oldhatnak meg a hallgatók.

A javasolt módszertan egyik apró elemeként különös hangsúlyt fektettem *A mesterséges intelligencia alapjai* című tantárgy oktatására. A tárgy sikerének záloga a Debreceni Egyetem MI-oktatásában kikristályosodott, pontos matematikai alapokon nyugvó „állapottér" szemlélet, amely az általam kidolgozott mintaprogramok elkészítését is befolyásolta. A tantárgy tematikája magában foglalja az egyszereplős problémák és a kétszemélyes játékok állapottér-reprezentációját, valamint számos algoritmust állapottér-reprezentált problémák megoldásának, illetve kétszemélyes játékok adott állapotában a legjobb lépésnek a keresésére [5, 6, 14, 11]. Javaslom a multiparadigmás nyelvek bevezetését a gyakorlati mesterséges intelligenciával foglalkozó tantárgyak oktatásába. Ha a kereső algoritmusok több, különböző megközelítést használó implementációját adjuk a hallgatók kezébe, az segíthet számukra megérteni az algoritmusok logikáját. Részletes példákat adok ilyen implementációkra C# és F# nyelven, amelyeket össze is hasonlítok egymással.

A programozó versenyek is kitűnő motivációt jelenthetnek a hallgatóknak, hogy megtanulják egy programozási nyelv eszközeinek, az alapvető és haladó adatszerkezeteknek vagy az őket feldolgozó alapvető és haladó algoritmusoknak a használatát. Két kollégámmal közösen kifejlesztettünk egy

webalkalmazást programozó versenyek lebonyolításának, valamint a hallgatók által különböző programozási feladatokra beküldött forráskódok kiértékelésének a támogatására. Röviden bemutatom ezt az alkalmazást, továbbá ismertetem az előnyeit más hasonló szoftverekkel szemben, valamint a vele szerzett tapasztalatainkat is.

A fentieket összefoglalva, dolgozatom megírásakor célkitűzéseim a következők voltak:

- néhány hosszú távú mintaprojekt megtervezése, amelyek az új módszertan keretein belül a hallgatókhoz rendelhetők;
- különböző mintaimplementációk készítése a mesterséges intelligencia legnépszerűbb kereső algoritmusaihoz C# és F# nyelven, köztük a kétszemélyes játékok esetében használt lépésajánló algoritmusokhoz is;
- az algoritmusok különböző implementációinak összehasonlítása (a tisztán objektumorientálttól a jórészt funkcionálisig);
- néhány példa bemutatása a fent említett algoritmusokhoz illeszthető konkrét problémák és játékok C# és F# implementációinak az elkészítésére;
- a multiparadigmás programozási nyelveknek a mesterséges intelligencia oktatásában történő felhasználásának az alátámasztása a fenti implementációk alapján;
- a Debreceni Egyetem Informatikai Kara által szervezett programozó versenyekkel, valamint a támogatásukra szolgáló alkalmazásokkal kapcsolatos tapasztalataink megosztása.

5. FEJEZET

Új eredmények

5.1. Új módszertan az informatikaoktatásban

Kutatásaim egyik fő területe az volt, hogy megpróbáltam választ találni az alábbi kérdésre: „Miért okoz problémát a legtöbb programtervező informatikus BSc szakos hallgatónknak a legtöbb haladó tantárgy követelményeinek a teljesítése?" A Debreceni Egyetemen szerzett tízéves oktatási tapasztalatom szerint hallgatóinknak számos nehézséggel kell szembenézniük tanulmányaik során. Ez részben a hallgatók nagy száma miatt van. Egyrészt sok kurzust kell indítanunk az egyes tárgyak gyakorlataiból, amelyeken így is túl sok a hallgató. Emiatt sok gyakorlatvezetőre (akár demonstrátorra) van szükség, akiknek sok hallgatóval kell foglalkozniuk, ezért sokkal kevesebb idő jut arra, hogy egyénenként foglalkozzanak velük. Másrészt sok hallgató nem az informatika iránti elkötelezettsége miatt jelentkezik az egyetemünkre, hanem más okokból (például szülői nyomás, az IT szakma népszerűsége, a jó elhelyezkedési lehetőségek vagy egyszerűen a szak céljainak félreértése miatt), és ebből fakadóan gyakran alulmotiváltak.

A tömegképzés azonban nem az egyetlen oka a „tömeges bukásnak" és a rossz teljesítménynek. Úgy vélem, hogy nekünk, az oktatóknak is van némi befolyásunk az oktatás eredményességére. A kulcs az, hogy találnunk kell egy módszert a hallgatók érdeklődésének a felkeltésére. Ennek egyik módja, hogy olyan feladatokat tűzünk ki számukra, amely érdekli őket. Ilyen feladat lehet például egy grafikus felülettel ellátott kétszemélyes játék elkészítése versenyképes mesterséges intelligenciával, egy könyvtári információs rendszer megírása, amely nyilvántartja a könyvek, olvasók és kölcsönzések adatait, vagy egy webalapú hálózatelemző segédprogram készítése, amely különböző statisztikai adatokat számít ki a hálózati forgalomról. Példaként olvashatunk a *Reversi* játék oktatási eszközként való felhasználásáról a [18] cikkben, az alapvető programozási fogalmak kétdimenziós játékok fejlesztésén keresztül történő oktatásáról a [10] tanulmányban, vagy az in-

formatika alapfogalmainak és a problémamegoldó stratégiáknak a diszkrét játékok fizikai és virtuális modelljeinek segítségével történő oktatásáról az [1] cikkben. Konkrétan a logikai kifejezések és a rekurzió fogalmának a tanításához két számítógépes játékot fejlesztett a szerző a [4] dolgozatban. Még a matematikai logika absztrakt tételeit is be lehet mutatni játékos feladatokon keresztül [17]. Manapság szinte lehetetlen olyan neves, informatikaoktatással foglalkozó folyóiratot vagy konferenciát találni, amely ne tartalmazna legalább egy, valós világból vett projekteken keresztül történő oktatásról szóló közleményt. Példaként említhetjük a *Computers & Education* (impakt faktor: 2,621, kiadó: Elsevier) vagy a *Journal of Computer Assisted Learning* (impakt faktor: 1,464, kiadó: Wiley) folyóiratokat, illetve az olyan konferenciákat, mint a *3rd Annual International Conference on Computer Science Education: Innovation and Technology* (amely 2012. november 19–20-án került megrendezésre Szingapúrban) vagy a *Consortium for Computing Sciences in Colleges — Northeastern Region* (amelyre 2013. április 12–13-án került sor a New York állambeli Albanyban).

Bármiről szóljon is a feladat, a lényeg, hogy egy nagyobb projekt legyen, azaz több olyan tárgyat lefedjen, amellyel a hallgató találkozik a tanulmányai során (de lefedheti akár a legtöbb kötelező tárgyat is). A projektmunka során alkalmazásra kerülnek a projekthez kötődő tárgyak keretein belül tárgyalásra kerülő ismeretek. A hallgatók megismerik a tárgyalt témakörök alkalmazhatóságát, hasznosságát, valamint az alkalmazás során felmerülő problémákat. Azt gondolom, hogy ha megfelelő feladatokat tudunk találni, akkor jobb teljesítményt érhetünk el nemcsak a kiadott feladatok megoldásában, hanem a kurzusok vizsgái során is.

Ha vetünk egy pillantást a Debreceni Egyetem programtervező informatikus BSc szakának kötelező és kötelezően választható tantárgyainak listájára [3], hamar észrevehetjük, hogy még egy közepes méretű szoftverfejlesztési projekthez is szükség van legalább az alábbi három kötelező tárgyon tanult ismeretekre: *Bevezetés az informatikába*, *Adatszerkezetek és algoritmusok* és *Magas szintű programozási nyelvek 1.* Egy hosszú távú projektben azonban a legtöbb kötelező tárgy érintett lehet. Könnyen találhatunk olyan izgalmas, valamely valós világbeli alkalmazáshoz hasonló feladatokat, amelyek hosszú távú projektként szolgálhatnak. Ha már meghatároztuk a projekt célját, nincs más hátra, mint olyan feladatokat találni, amelyeket felhasználhatunk a kapcsolódó tantárgyak oktatásában. Véleményem szerint ha a hallgatók olyan összetett projekten dolgoznak, amelynek a fejlesztési folyamata felöleli a tanulmányaik csaknem teljes időtartamát,

akkor motiváltabbak lesznek, és jobban fogják látni az egyes kurzusokon megtanult ismeretek közötti kapcsolatot.

Valós világbeli alkalmazások fejlesztéséhez kötődő feladatok kitűzése a következő előnyökkel jár:

- Az érdelődést felkeltő példák növelik a hallgatók motiváltságát.
- Egy összetett projekten keresztül a hallgatók az informatika számos területét gyakorolhatják.
- Ha több kurzuson is ugyanazzal az egy nagyobb projekttel találkoznak, az segít a hallgatóknak jobban megérteni a különböző kurzusok mögött rejlő ismeretek közötti összefüggéseket.
- A projektek gyakorlatorientáltabbá teszik az informatikaoktatást.
- A projektek hitelessé teszik az előadások során elsajátított ismereteket, és választ adnak arra a kérdésre is, hogy hogyan alkalmazzuk ezt a tudást.

Szeretném hangsúlyozni, hogy a projektorientált oktatás ötletét a magyar egyetemek legtöbb informatikai mesterképzésén már alkalmazzák. A jelenleg hatályos felsőoktatási törvény szerint még az alapképzéseknek is kell tartalmazniuk valamennyi projektmunkát. Kitűnő példája ennek az Eötvös Loránd Tudományegyetem Informatikai Kara, ahol a hallgatók részt vehetnek egy úgynevezett kooperatív képzésben 16 kreditért [2]. A kooperatív képzés célja, hogy a hallgatóknak lehetőséget biztosítson arra, hogy közelebbről is megismerkedhessenek az informatikus szakma gyakorlati oldalával valós informatikai cégek gyakorlott szakembereinek az irányításával. Egy másik példa az *Önálló laboratórium* című tárgy a Budapesti Műszaki és Gazdaságtudományi Egyetem Villamosmérnöki és Informatikai Karán [13], illetve a Debreceni Egyetem Informatikai Karán, amelynek keretében a hallgatók elmélyíthetik ismereteiket és tapasztalatokat szerezhetnek az informatika egy bizonyos területén. Külföldi intézmények is alkalmazzák a projektalapú oktatást; a Paderborni Egyetemen például háromfős csoportoknak kell operációs rendszert fejleszteniük az egyik BSc-s kurzus keretében.

Van azonban néhány fontos különbség a javasolt projektalapú megközelítés és a fent említett példák között:

- Mind a kooperatív képzés, mind az *Önálló laboratórium* tárgyak függetlenek az alapképzések kötelező tárgyaitól abban az értelemben,

hogy különálló oktatási egységnek számítanak. A javaslatom szerint a projektek a legtöbb kötelező tárgy oktatási anyagának szerves részét képeznék, így a hallgatók a már meglévő tárgyak kurzusain dolgozhatnak a projekteken, nincs szükség tehát újabb tárgyakra vagy képzésekre.

- Míg a kooperatív képzés és az *Önálló laboratórium* tárgyak az informatikának csak egy-két területére koncentrálnak, addig a javasolt megközelítés a legtöbb kötelező tárgy tematikáját felöleli.

- Mind a kooperatív képzésnek, mind az *Önálló laboratórium* tárgyaknak szigorú előfeltételei vannak, azaz a korábbi tanulmányok alatt elsajátított ismeretekre épülnek. A javasolt megközelítés használatával azonban a hallgatók már az első félév folyamán elkezdenek projekteken dolgozni. Ez azt is jelenti, hogy az oktatóknak sokkal több hallgatóval kell foglalkozniuk, akik korábban még nem vettek részt projektmunkában. Másrészt az oktatók sem rendelkeznek feltétlenül nagy tapasztalatokkal a projektmenedzsment területén, és együtt kell működniük egymással a jobb eredmény elérése érdekében.

- Bár a kooperatív képzés az oktatás részét képezi, az intézmény elveszíti a jogát a képzés és a számonkérések teljes körű kézben tartására. További hátránya, hogy a fővároson kívül nem olyan egyszerű a szükséges számban olyan céget találni, amely megfelelő projektekkel tud szolgálni.

1. tézis: Kidolgoztam egy újszerű módszertant az informatikai alapképzések oktatásához, amely hosszú távú szoftverfejlesztési projektek bevezetését jelentené a legtöbb kötelező tárgy gyakorlatainak oktatásába, izgalmas valós világbeli alkalmazásokat eredményezve. Bár az oktatók részéről szükséges némi pluszmunka, a hallgatók motiváltabbak lennének, és jobban átlátnák a különböző kurzusokon megtanult ismeretek közötti összefüggéseket. Ismertettem néhány mintaprojektet is, a hozzájuk kapcsolódó kötelező tárgyak gyakorlati kurzusain alkalmazható mintafeladatokkal együtt.

5.2. Multiparadigmás programozási nyelvek használata a mesterséges intelligencia oktatásában

Egészen pár évvel ezelőttig a programozás tiszta, egyparadigmás technikák használatáról szólt. Manapság azonban a programozási nyelvek „konvergálnak" egymáshoz, azaz az egyik paradigma eszközei megjelennek olyan nyelvekben is, amelyek egy másik paradigmára épülnek [15, 16]. Ezt a folyamatot elsősorban az imperatív, objektumorientált és funkcionális programozási nyelvek területén figyelhetjük meg. A C# vagy a D például az imperatív C nyelven alapulnak, kiegészítve objektumorientált és funkcionális nyelvi elemekkel. Egy másik példa a CLOS, amely egy, a Lispre épülő funkcionális nyelv hozzáadott objektumorientált eszközökkel. Az ilyen nyelveket *multiparadigmás programozási nyelveknek* nevezzük.

Ezeknek a nyelveknek az a legnagyobb előnye, hogy a programozó kiválaszthatja azt a paradigmát, amely a legalkalmasabb egy konkrét feladat megoldására. Akár azt is megtehetjük, hogy ugyanazon program különböző részeit különböző programozási paradigmákból vett technikák használatával készítjük el. Ha egy nagyobb alkalmazás egy kisebb része nagy hatékonyságot igényel, imperatív kódot használunk, az újrafelhasználható típusokat objektumorientáltan hozzuk létre, míg bizonyos problémakör esetén a funkcionális megközelítést alkalmazzuk, mert az ebbe a körbe tartozó problémák megoldásai így tömörebben kifejezhetők, mint imperatív kód használatával.

Úgy vélem, hogy a multiparadigmás nyelvek nagy segítséget jelenthetnek az informatika oktatásában. Az oktatók ugyanannak az algoritmusnak különböző változatait mutathatják be a hallgatóknak az implementálásához felhasznált paradigmá(k)tól függően, a hallgatók pedig később eldönthetik, melyik paradigmá(ka)t használják egy házi feladat, vizsga vagy munkahelyi feladat esetén a probléma specifikációjától és a saját programozási előismereteiktől függően. A projektszemléletű módszertan kiválóan alkalmas az egyes paradigmák előnyeinek az illusztrálására. A hallgatók több paradigma mentén is megoldhatják ugyanazt a feladatot, összehasonlíthatják a különböző paradigmák lehetőségeit és korlátait a kapott kódok alapján, amelyekből aztán összeválogathatják a legelőnyösebb tulajdonságú részeket. Természetesen erre a célra nem feltétlenül kell multiparadigmás nyelvet használnunk, használhatunk több egyparadigmás nyelvet is, de így nem kell megtanítani, a hallgatóknak pedig megtanulni még egy újabb programozási nyelvet csupán egy adott algoritmus kódolásáért.

Kutatásaim során megvizsgáltam az F# mint egy új multiparadigmás programozási nyelv használatának lehetőségét a mesterséges intelligencia (MI) területén használt különféle algoritmusok kódolására. Három fő okból választottam ezt a területet. Egyrészt korábban oktatója voltam *A mesterséges intelligencia alapjai* című tárgy gyakorlatainak a Debreceni Egyetemen [19]. Másrészt az MI, azon belül konkrétan a kereső algoritmusok olyan számítási területet képviselnek, amely esetén jól alkalmazható a funkcionális programozás, mivel ezeknek az algoritmusoknak bizonyos részei (mint például az operátoralkalmazási előfeltételek ellenőrzése) alapvetően funkcionálisak. Harmadrészt rendelkezésemre állt a kérdéses algoritmusok néhány imperatív és objektumorientált implementációja, amelyek jó kiindulópontnak bizonyultak az F# verzió elkészítéséhez [8, 7].

Nem nehéz belátni, hogy az MI problémák megoldására a funkcionális programozás előnyösebb, mint az imperatív programozás. Ahogy a [12] cikkemben bemutattam, még az egyszerű 8 királynő problémának is sokkal tömörebb megoldását tudjuk megadni F#-ban (kizárólag funkcionális programozási elemek felhasználásával), mint C-ben vagy C#-ban. Ha azonban a hatékonyság vagy az újrafelhasználhatóság is szerepet játszik, nem biztos, hogy a tisztán funkcionális kód lesz a legjobb megoldás bizonyos problémák esetén.

Egy-egy objektumorientált C# implementáción kívül elkészítettem három F# implementációt is az egyszereplős problémák kereső algoritmusaihoz, illetve kettőt a kétszemélyes játékok lépésajánló algoritmusaihoz. A célom az volt a különböző implementációk elkészítésével, hogy több megközelítést biztosítsunk a hallgatók számára ugyanazon pszeudokódok megértéséhez. Bár azt nem tudom, hogy funkcionálisan gondolkodva könnyebben megértik-e az algoritmusok működését, egynél több implementáció megismerése biztosan nem árthat. Íme egy különböző szempontokon alapuló, részben szubjektív összehasonlítása az egyszereplős problémák esetén használatos kereső algoritmusok négyféle implementációjának:

- *Forráskód metrikák*: Annak ellenére, hogy funkcionális nyelvek esetén nem alkalmazhatók ugyanazok a metrikák, mint imperatív nyelvek esetén, most három olyan metrikát fogok használni, amelyek mind C#-ban, mind F#-ban relevánsak lehetnek: a sorok száma, az osztályok száma és a függvények száma (beleértve a metódusokat is). Az 5.1. táblázat összefoglalja ezeknek a metrikáknak az értékeit a különböző implementációk esetén.

  A *sorok száma* metrika az összes forrásállományban szereplő kód-

	C#	F# v1	F# v2	F# v3
Sorok száma	842	537	536	417
Osztályok száma	15	13	6	4
Függvények száma	49	43	43	34

5.1. táblázat. Néhány forráskód metrika.

sorok számát adja meg, beleértve az üres sorokat is, nem beleértve viszont a konkrét problémákhoz tartozó forráskódokat. Az *osztályok száma* a forráskódban definiált osztályokra vonatkozik, kivéve a kivétel osztályokat, a felsorolásos típusokat és az `IComparer` osztályokat. A *függvények száma* tartalmazza az összes függvényt és konkrét metódusimplementációt, beleértve a konstruktorokat, nem beleértve viszont a lambda-kifejezéseket.

Ahogy látható, a harmadik F# implementáció feleakkora méretű, mint a C# implementáció. Ez a különbség persze főleg az F# nyelv kompakt szintaxisából következik. A másik oka annak, hogy az F# implementációk rövidebbek, az az, hogy hiányzik belőlük két osztály a C# kódból, amelyek 57 sort tesznek ki.

Az osztályok számának csökkenése az egyre „funkcionálisabb" kód eredménye. Az első F# implementáció – ahogy fent említettem – két osztállyal kevesebbet tartalmaz, mint a C# verzió. A második változatban a hét konkrét algoritmus osztályt hét függvénnyel helyettesítettük. A harmadik változatban a két absztrakt kereső osztályt is függvénnyé konvertáltuk. Ha tisztán funkcionális kódot szeretnénk, a maradék osztályoktól is meg kellene szabadulnunk, de az nem eredményezne rövidebb vagy olvashatóbb kódot. Az osztályok használatával könnyű *újrafelhasználható* kódot írni például az operátoralkalmazásra: csupán meg kell hívnunk az absztrakt `State` osztály `Apply` metódusát, anélkül hogy tudnánk, hogyan is implementálta azt valamely konkrét probléma állapotait reprezentáló konkrét osztály. A két `Node` osztály konstruktorai pedig pontosan ezt csinálják. A két `Node` osztályt valójában lecserélhetnénk rekord típusokra, mivel belőlük nem származnak más osztályok, de így sem kapnánk olvashatóbb kódot, ráadásul nem tudnánk használni az olyan .NET metódusokat, mint például a `Contains`.

A függvények száma nagyon hasonló az egyes implementációkban. Ez azért van, mert az osztályok metódusai egy-egy függvénynek felelnek meg az osztálymentes implementációkban; még a konstruktoroknak is függvényeket feleltetünk meg. A harmadik F# implementáció azonban valamivel kevesebb függvényt tartalmaz, mint a másik három. Ennek az az oka, hogy hiányzik belőle a `ToString` metódusnak a konkrét algoritmus osztályokban (illetve objektumokban) megtalálható hét implementációja. Ezeket mindössze egyetlen függvény, a `printSearchInfo` helyettesíti. Ugyanez érvényes a gráfkereső algoritmusok `Expand` és `Search` metódusaira. A három metódusnak megfelelő függvények mindegyike `match` kifejezéseket tartalmaz, amelyeknek az alapja a kereséshez használt algoritmus. Ez jól mutatja a különbséget annak a problémának az objektumorientált és funkcionális megközelítése között, hogy hogyan tudunk egy alaposztályhoz új alosztályt, illetve az alosztályokhoz új funkcionalitást bevezetni. Ha új alosztályt szeretnénk bevezetni, akkor az OO megközelítés jobb, mert nem kell hozzányúlnunk a már meglévő alosztályokhoz, csak meg kell írnunk az új osztályt az alaposztályból örökölt összes funkcionalitással. Funkcionális megközelítés esetén ezzel szemben az összes `match` kifejezéshez egy-egy új ágat kell hozzávennünk. Ha viszont új funkcionalitással szeretnénk bővíteni a már meglévő alosztályainkat, akkor a funkcionális módszer a jobb, hiszen csak egy új függvényt kell írnunk a meglévőkhöz hasonló `match` kifejezéssel. OO megközelítést használva az alaposztályt ki kell egészítenünk egy új metódussal, valamint az összes alosztályt az új metódus egy-egy implementációjával.

- *A felhasznált módosítható adatszerkezetek*: Ebben a tekintetben nincs különbség az implementációk között. Bár a tisztán funkcionális programok egyáltalán nem alkalmaznak módosítható adatokat, mindegyik implementációban használtam néhány módosítható adatszerkezetet. Ha szeretnénk, ezen adatkollekciók bármelyikének a típusát lecserélhetnénk az F# nem módosítható `list` vagy `seq` adattípusára. Az így kapott kód ugyanolyan méretű lenne, viszont kevésbé hatékony, mivel a `List` és `Seq` modulok rekurzív függvényei lassabbak, mint a megfelelő .NET metódusok. Ez különösen akkor igaz, amikor elemeket adunk hozzá ezekhez a kollekciókhoz, vagy amikor elemeket törlünk belőlük: egy nem módosítható adatszerkezet esetén le kell másolnunk az eredeti kollekciót, egy apró módosítást végrehajtva az elemein. Ez az oka annak, hogy .NET kollekciókat használtam az

F# nem módosítható adattípusai helyett a kérdéses adatszerkezetek tárolására.

- *A felhasznált funkcionális nyelvi eszközök*: A C# implementáció egyetlen funkcionális elemet sem tartalmaz, tisztán objektumorientált. Az első F# implementáció farokrekurzív függvényekkel és szekvenciaműveletekkel helyettesíti a ciklusokat. Bár az objektumkifejezések nem funkcionális nyelvi eszközök, a második F# implementáció sokat tartalmaz belőlük az alosztályok kiváltására. A harmadik F# implementáció tartalmazza a legtöbb funkcionális elemet: diszkriminált uniót használunk az algoritmus típusának tárolására, `match` kifejezéseket a kezelésére, néhány függvényt első osztályú értékként használunk, és van benne egy magasabb rendű függvény is.

- *Hatékonyság*: Mivel a funkcionális nyelvek absztraktabbak, mint az objektumorientált nyelvek, összetettebb futtató rendszerre van szükségük. Ez a fő oka annak, hogy a funkcionális programok kevésbé hatékonyak, még akkor is, ha lefordítjuk, nem pedig interpretáljuk őket. Lefuttattam az összes implementáció bemutatott főprogramjait ugyanazon a számítógépen, egy Gigabyte T1018X TouchNote netbookon Intel Atom N280 processzorral 1,33 GHz-en és 1 GB memóriával, az összes programot Microsoft Visual Studio 2010-ben fordítva: a C# program kevesebb, mint fél másodperc alatt végzett, míg az F# programok mindegyike nagyjából 7 másodpercig futott. Ahogy írtam, ennél is rosszabb eredményt kaptunk volna, ha az F# nem módosítható adattípusait használtuk volna.

Az egyszereplős problémák esetén alkalmazott algoritmusok különböző implementációira vonatkozó megfigyelések a kétszemélyes játékok lépésajánló algoritmusaira is érvényesek, bár a különbség az utóbbi esetben kisebb. Ugyanez mondható el a konkrét problémák és játékok implementációival kapcsolatban is.

Végső konklúzióként az a véleményem, hogy nem érdemes az egyik vagy a másik paradigmához ragaszkodni, ha többet is használhatunk egy programon belül. A funkcionális kód néha absztraktabb, olvashatóbb vagy egyszerűen csak rövidebb, mint az objektumorientált megfelelője. Másrészről az OO kód általában hatékonyabb és néha újrafelhasználhatóbb, mint a funkcionális megfelelője. Ezért azt gondolom, hogy a multiparadigmás nyelvek, mint amilyen az F#, előnyösebbek lehetnek főleg nagyobb szabású alkalmazások esetén, de akár kisebb programokban is. A hallgatók

különbözőek, így néhányuk jobban megértheti az algoritmusokat egy funkcionálisabb megközelítésre, mint egy tisztán objektumorientált kódra alapozva. Például a `Seq.fold` függvény egy akkumulátor paraméterrel ellátott lambda-kifejezéssel jobban leírhatja egy minimum- vagy maximumkiválasztás működését az olyan hallgatók számára, akik rekurzívan (funkcionálisan) gondolkodnak.

> **2. tézis:** Módszertani útmutatást adtam *A mesterséges intelligencia alapjai* című tárgy gyakorlatainak oktatásához, amely a legnépszerűbb kereső algoritmusok, valamint egy-két konkrét probléma és kétszemélyes játék néhány lehetséges C# és F# nyelvű implementációját tartalmazza. Javaslom a multiparadigmás programozási nyelvek használatát ezeken a kurzusokon, mert nem létezik egy olyan paradigma, amely minden probléma megoldására egyformán alkalmas lenne, és mert az oktatók (ahogy később a hallgatók is) kiválaszthatják a számukra legmegfelelőbb paradigmát valamely probléma egy konkrét részének a megoldására.

5.3. Programozó versenyek lebonyolítása a ProgCont alkalmazással

A programozó versenyek erős motivációs tényezőt jelenthetnek. Több mint tíz éve rendezünk félévenként legalább egy házi versenyt, és elmondhatom, hogy legalább néhány hallgató érdeklődését felkeltik ezek a versenyek. A tíz év alatt két alkalmazást is kifejlesztettünk a versenyzők által beküldött megoldások on-line kiértékelésére. Az első neve *Programming Contest Result Manager* (PCRM), és egy e-mail-alapú konzolos alkalmazás, a másodikat pedig *ProgCont*-nak hívják, amely egy kliens/szerver architektúrájú webalkalmazás. A PCRM-et részletesen tárgyalják a [9, 7] tanulmányok, most a ProgContot mutatom be a vele szerzett tapasztalatainkkal együtt.

A tanulmányi versenynek az a szerepe az oktatásban, hogy motiválja a résztvevőt egy-egy témakör mélyebb megismerésére, olyan egyéni célkitűzést jelent a diákok számára, amely önálló munkára sarkall. Amellett, hogy a versenyszellem ösztönzően hat, pozitív hatást fejt ki a tanulmányi eredményre; egy nívós versenyen való részvétel, egy-két elért jó eredmény szakmai tapasztalatként is kamatoztatható. Továbbá egy-egy tanulmányi verseny szakmai kapcsolatok kiépítésében is segítséget nyújthat.

A Debreceni Egyetem hallgatói 1995 óta vesznek részt az ACM Nemzetközi Felsőoktatási Programozó Verseny (ACM International Collegiate Programming Contest) közép-európai selejtezőiben (bár 1998 és 2000 között az egyetem nem nevezett a versenyre). Másodéves hallgatóként magam is tagja lehettem annak a csapatnak, amely 1995-ben továbbjutott a helyi fordulóból a regionális fordulóba. 2001-től szervezőként és zsűritagként veszek részt az ACM ICPC és más programozó versenyek helyi fordulóiban.

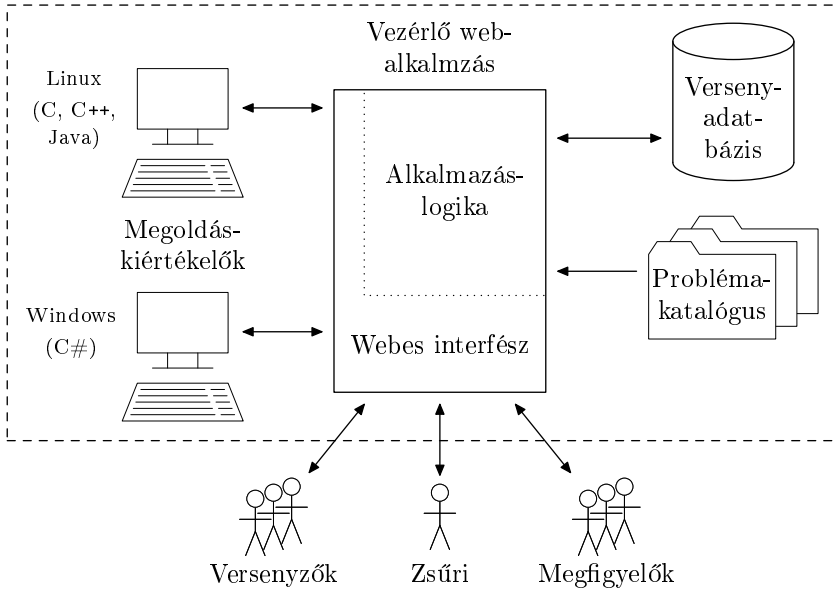
A helyi egyetemi fordulók lebonyolítását a kezdeti időkben igencsak megnehezítette, hogy a versenyzők által beküldött megoldások ellenőrzését kézzel kellett elvégeznünk, ami a kényelmetlenségeken túl a zsűri hatékony munkáját is akadályozta, és rengeteg hibalehetőséget rejtett magában. E problémák kiküszöbölésére határoztuk el, hogy elkészítünk egy olyan szoftvert, amely nagy mennyiségű megoldást képes feldolgozni versenykörülmények között és off-line módban egyaránt. Kósa Márk kollégámmal és egy rátermett hallgatóval, Gunda Lénárddal közösen 2004-ben kifejlesztettünk egy e-mail-alapú konzolos alkalmazást, a *Programming Contest Result Manager*-t (PCRM), amely nemcsak a versenyek alatt beküldött megoldások kiértékelésében segített, hanem az olyan konkrét tantárgyak házi feladatainak ellenőrzésében is, mint *A mesterséges intelligencia alapjai*.

A PCRM-et használtuk pár éven keresztül, de később már nem felelt meg az egyre újabb elvárásainknak. Szerettünk volna például egy felhasználóbarát grafikus felületet az alkalmazáshoz, mégpedig inkább webes felületet mind a versenyzők, mind a zsűri számára. A másik hátránya az volt, hogy a beküldött megoldásokat POP3 protokollal töltötte le, és ez nehézkessé tette a használatát. Úgy döntöttünk, hogy nem dolgozzuk át a meglévő alkalmazásunkat, hanem tesztelünk néhány mások által készített programot (abban az időben már találtunk pár programot az interneten, amelyekről úgy tűnt, hogy megfelelnek az igényeinknek), valamint egyúttal kifejlesztetünk tehetséges hallgatókkal egy vadonatúj webalkalmazást szakdolgozat keretében. Utóbbi eredményeképpen 2009-ben megszületett egy ASP.NET-alapú webalkalmazás, amelyet egy hallgató készített, és egészen jól használható volt, de akadt benne pár hiba, és miután a hallgató elhagyta az intézményünket, nem volt, aki karbantartsa. A legjobb külső program, amit találtunk és teszteltünk, a PC^2 (Programming Contest Control System) volt, amelyet a Kaliforniai Állami Egyetemen fejlesztettek, de abból is hiányzott néhány szükséges funkció.

Így született meg 2011-ben a ProgCont, amelyet Kádek Tamással és Kósa Márkkal közösen hoztunk létre. Mindeddig ez a legjobban használ-

ható és legrobusztusabb alkalmazás programozó versenyek támogatására, amellyel valaha találkoztunk. Az alkalmazás folyamatos fejlesztés alatt áll; legutóbb azért kellett kiegészítenünk, hogy támogasson egy új típusú versenyt, az Informatikai Kar Regionális Programozó Csapatversenyét, amelyet első alkalommal rendeztünk meg 2012 novemberében öt környező megye középiskolái és főiskolái számára, és amely az ACM jellegű versenyektől némileg eltérő kiértékelési rendszert használt.

A ProgCont rendszer négy, egymástól jól elkülöníthető összetevőből épül fel: problémakatalógus, versenyadatbázis, vezérlő webalkalmazás és megoldáskiértékelő kliensek (lásd az 5.1. ábrát).



5.1. ábra. A ProgCont rendszer felépítése.

A *problémakatalógus* tartalmazza a később összeállított versenyekre szánt anyagot, a problémák – esetenként többnyelvű – leírását a hozzá tartozó ábrákkal és további (akár letölthető) segédanyagokkal együtt, valamint szintén itt találhatók a megoldások ellenőrzéséhez szükséges tesztesetek is. Egy-egy problémához több teszteseteket tartalmazó állomány is készíthető. A megoldások helyességének ellenőrzése az alábbi módokon történhet:

a program ellenőrizheti, hogy a beküldött megoldás által előállított kimenet karakterenként megegyezik-e egy előre generált kimenettel, illetve a kimenet helyessége vizsgálható külső programmal. Tesztesetenként határozható meg a kimenet előállításának időkorlátja is. A problémakatalógus az operációs rendszer könyvtár- és fájlszerkezetének segítségével alakítható ki. Minden probléma külön könyvtárban kap helyet. A probléma szövege egy XML fájlban, a hozzá tartozó tesztesetek, az előre generált kimeneti állományok és a tesztelési paraméterek pedig egy tömörített (ZIP) fájlban tárolódnak. A szöveges leírásban hivatkozott további dokumentumok szintén ugyanebben a könyvtárban kapnak helyet.

A *versenyadatbázis* írja le a versenyek, a feladatok, a versenyzők, a megoldások és a kiértékelések kapcsolatát. Egy-egy verseny a problémakatalógusból válogatott feladatok, csapatverseny esetében a csapatokat alkotó versenyzők, valamint a verseny lebonyolítására vonatkozó technikai adatok (például a verseny kezdetének és befejezésének az ideje vagy az egyes feladatok esetében engedélyezett programozási nyelvek) összessége. Az adatbázis tárolja a versenyzők által beküldött megoldásokat, majd a megoldások értékelésének az eredményét, amelyeket egy-egy megoldáskiértékelő kliens szolgáltat. Mindezeket a ProgCont alkalmazásban egy PostgreSQL adatbázisban tároljuk.

Egy-egy verseny lebonyolítását a *vezérlő webalkalmazás* irányítja. A versenyzők a webalkalmazás segítségével böngészhetik a feladatokat, tölthetik fel az egyes feladatok megoldásait, és értesülhetnek a megoldáskiértékelések eredményéről és a verseny állásáról. A megfigyelők (például a versenyzők felkészítői vagy a vendégek) szintén követhetik az aktuális eredménylistát. A zsűri a webes felületen tudja beállítani a verseny bizonyos paramétereit, például azt, hogy az egyes feladatokra milyen programozási nyelven várja a megoldásokat, hogy hány pontot érnek az egyes feladatok, vagy hogy mennyi büntetőidő jár egy-egy rossz megoldásért. A webalkalmazás ütemezi a beküldött megoldások kiértékelésének sorrendjét is, ami azt jelenti, hogy ő osztja ki azokat az egyes, a rendszerhez éppen csatlakozó megoldáskiértékelő kliensek számára. A programkódok kiosztása a kliensek által küldött azon információ alapján történik, hogy milyen programnyelvű kódok kezelésére alkalmasak. A vezérlő webalkalmazás futtatására Apache Tomcat webalkalmazás-szervert használunk. A webalkalmazás és a felhasználók (a versenyzők és a zsűri) közötti kommunikáció biztonságát az SSL (HTTPS) protokoll szavatolja. A webalkalmazás JDBC-n keresztül éri el a versenyadatbázist.

A *megoldáskiértékelő kliensek* olyan önálló alkalmazások, amelyek periodikusan (5 másodpercenként) ellenőrzik, hogy van-e olyan kiértékelésre várakozó megoldás, amelyet kezelni tudnak. Ha van, a kliens első lépésben megpróbálja lefordítani a vezérlő webalkalmazás által neki kiosztott programkódot. Amennyiben a fordítás sikeres, lefuttatja a programot minden tesztesetre egy előre beállított futtatási környezetben. Az aktuális teszteseteket a problémakatalógusból tölti le a webalkalmazáson keresztül, valahányszor megváltozik az őket tartalmazó ZIP állomány (és persze az első alkalommal, amikor szükség van rájuk). A tesztesetek kiértékelésekor a megoldáskiértékelő kliens figyelembe veszi a tesztesetre beállított maximális futási időt. Ha a program időben megáll, a kliens ellenőrzi a kimenetét az előre beállított módszernek megfelően: vagy összehasonlítja a letöltött kimeneti állománnyal, vagy továbbítja a külső kiértékelő programnak. Az összesített eredményt végül továbbítja a vezérlő webalkalmazás felé.

Mivel az egyes programozási nyelvekhez különböző operációs rendszerek passzolnak inkább, a megoldáskiértékelő klienseket platformfüggetlen Java implementációval készítettük. Így lehetőség van arra, hogy a vezérlő webalkalmazástól és a többi kiértékelő klienstől különböző (lehetőleg virtuális) számítógépeken fussanak, ezáltal nem veszélyeztetve azok működését, amennyiben egy kártékony kód tönkretenné egy konkrét kliens futtatási környezetét. A C# kódokat például fordíthatja és futtathatja egy Windows-alapú kliens, míg a C, C++, Java és Pascal kódokhoz egy másik, Linux-alapú klienst használhatunk. Minél több megoldáskiértékelő klienst használunk, annál több kiértékelést hajthatunk végre egyidőben.

2011. október 2-án rendeztük az első olyan versenyt, amelyet a ProgCont alkalmazással vezéreltünk. Az eredeti célunknak megfelelően ez az adott év ACM versenyének helyi fordulója volt. Kihasználva a versenyek paraméterezhetőségének a lehetőségét, 2012. február 6-án egy rövid egyéni versenyre került sor az *Informatikai versenyfeladatok* című kurzus hallgatói között. Ugyanabban a félévben három különböző tantárgy ismereteinek az elmélyítésében is tudtunk segíteni a hallgatóknak olyan versenyekkel, amelyek több mint egy hónapig tartottak: *Informatikai versenyfeladatok*, *Magas szintű programozási nyelvek 1* és *A mesterséges intelligencia alapjai*. Ezután egy ACM jellegű versenyt szerveztünk 2012. május 6-án (amely a marosvásárhelyi ECN Nemzetközi Programozó Verseny selejtezője volt), egy újabb helyi ACM versenyt 2012. október 7-én, végül, de nem utolsósorban pedig az Informatikai Kar Regionális Programozó Csapatversenyét 2012. november 25-én.

3. tézis: A programozó versenyek további motivációs tényezőt jelentenek a hallgatók számára. Két kollégámmal közösen lefektettük egy versenykezelő szoftver elvi alapjait, majd megterveztünk és kifejlesztettünk egy webalkalmazást, amely ACM jellegű és egyéb típusú versenyek lebonyolításának támogatására, valamint a különböző kurzusokon kiadott házi feladatok megoldásaiként beküldött forráskódok kiértékelésére egyaránt alkalmas. A *ProgCont* egy platformfüggetlen, paraméterezhető versenykezelő rendszer, amely jól beilleszthető a javasolt projektalapú oktatási módszertanba.

6. FEJEZET

Összefoglalás

Összefoglalva az eredményeimet, remélem, sikerült elkészítenem néhány olyan eszközt, amelyek megkönnyíthetik a hallgatók életét a tanulmányaik során. Az egyik ilyen eszköz az lehet, hogy olyan hosszú távú projekteket jelölünk ki számukra, amelyeken már az első félévtől kezdve dolgozhatnak. Egy valós világbeli alkalmazás mindig izgalmasabb, mint a kis programrészletek, egy hosszú szoftverfejlesztési folyamat tehát növelheti a hallgatók tanulási és programozási hajlandóságát. Az alkalmazás megfelelő részeit a projekthez kötődő kurzusok keretein belül, az oktató irányítása mellett készíthetik el. A másik előnyös hatása ennek a megközelítésnek az, hogy a hallgatók jobban átlátják az összefüggéseket a különböző kurzusokon elsajátított ismeretek között. Két ilyen hosszú távú projektet ismertettem, a kapcsolódó tantárgyakhoz kötődő néhány lehetséges feladattal együtt.

Módszertani útmutatást adtam *A mesterséges intelligencia alapjai* című vagy egyéb, a mesterséges intelligenciához kapcsolódó tantárgy oktatásához. Ez egyszereplős problémák és kétszemélyes játékok esetén alkalmazható kereső algoritmusok különböző C# és F# nyelvű implementációiból áll, amelyek között a különbség abban rejlik, hogy mennyi funkcionális programozási elemet tartalmaznak. Elkészítettem néhány konkrét probléma és játék C# és pár F# implementációját is. A hallgatók és az oktatók kiválaszthatják közülük azt a változatot, amelyik a leginkább illik hozzájuk a gondolkodási módjuk és a programozási ismereteik alapján. Ezáltal a hallgatók könnyebben megérthetik az előadásokon ismertetett kereső algoritmusok működését. Az elkészült implementációk összehasonlításából azt a következtetést vontam le, hogy a multiparadigmás programozási nyelvek használata a mesterséges intelligencia oktatásában mind az oktatók, mind a hallgatók számára hasznos lehet.

A programozó versenyek is erős motivációs tényezőt jelenthetnek a hallgatók számára. Már több mint tíz éve szervezünk ACM jellegű versenyeket, és elmondhatom, hogy van jónéhány olyan hallgató, akik egyetlen alkalmat

sem szalasztanának el, hogy megmérettessék magukat másokkal egy-egy versenyen. Nemcsak az ő kedvükért, de szükségünk volt egy olyan alkalmazásra, amely ACM jellegű és esetleg más típusú programozó versenyek támogatására is képes. Mivel nem találtunk olyan versenykezelő szoftvert, amely minden igényünket kielégítette volna, két kollégámmal közösen úgy határoztunk, hogy kifejlesztünk egy vadonatúj webalkalmazást a programozó versenyek vezérlésére. A *ProgCont* rendszert úgy terveztük, hogy moduláris felépítésű és paraméterezhető legyen, valamint hogy könnyen bővíthető legyen új támogatott programozási nyelvekkel.

References

[1] Juan Albornoz Bueno and Raúl Alfredo 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*, 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.

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

[6] Iván Futó, editor. *Mesterséges intelligencia.* Aula, Budapest, 1999.

[7] Márk Kósa. Korszerű információtechnológiai módszerek bevezetése a mesterséges intelligencia oktatásába. Doctoral dissertation, University of Debrecen.

[8] Márk Kósa and János Pánovics. Keresőalgoritmusok objektumorientált megközelítése a Mesterséges intelligencia tárgy bevezető kurzusán.

Proceedings of the 17th International Conference on Computers and Education, pages 94–97, 2007.

[9] Márk Kósa, János Pánovics, and Lénárd Gunda. An evaluating tool for programming contests. *Teaching Mathematics and Computer Science*, 3(1):103–119, 2005.

[10] Scott T. Leutenegger and Jeffrey Edgington. A games first approach to teaching introductory programming. *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, 39(1):115–118, 2007.

[11] Bernd Owsnicki-Klewe. Search algorithms. http://users.informatik.haw-hamburg.de/~owsnicki/search.html.

[12] János Pánovics. A functional programming approach to AI search algorithms. *Journal of Information Technology Education: Innovations in Practice*, 11:353–376, 2012.

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

[14] Stuart J. Russell and Peter Norvig. *Mesterséges intelligencia modern megközelítésben*. Panem, Budapest, 2005.

[15] Michael L. Scott. *Programming language pragmatics*. Morgan Kaufmann, San Francisco, 2009.

[16] Robert W. Sebesta. *Concepts of programming languages*. Pearson Education, New Jersey, 2012.

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

[18] D. W. Valentine. Playing around in the CS curriculum: Reversi as a teaching tool. *Journal of Computing Sciences in Colleges*, 20(5):214–222, 2005.

[19] Magda Várterész, Benedek Nagy, Márk Kósa, and János Pánovics. A Mesterséges intelligencia tárgy bevezető kurzusának gyakorlatai a Debreceni Egyetemen. *Informatika a Felsőoktatásban 2002*, pages 1103–1109, 2002.

List of Publications

Peer-Reviewed Papers in the Subject of the Dissertation

1. Kádek Tamás, **Pánovics János**: Extended breadth-first search algorithm, *International Journal of Computer Science Issues* (2013) **10** (6), No. 2, pp. 78–82. Impact Factor: 0.242.

2. **Pánovics János**: Motivating students with projects encompassing the whole duration of their studies, *Teaching Mathematics and Computer Science* (2013) **11** (2), pp. 165–180.

3. **Pánovics János**: A functional programming approach to AI search algorithms, *Journal of Information Technology Education: Innovations in Practice* (2012) **11**, pp. 353–376. Impact Factor: h-index: 28.

   Ref. number: Education Resources Information Center ERIC #EJ990988. (http://www.eric.ed.gov)

4. Kósa Márk, **Pánovics János**: Search algorithms at ACM contests, *Proceedings of the 7th International Conference on Applied Informatics (ICAI 2007)*, Eger, Hungary, January 28–31, 2007, Vol. II., pp. 367–375 (2009).

   Ref. number: Zentralblatt MATH Database Zbl 1183.68722. (http://www.zentralblatt-math.org/zbmath/)

5. Kósa Márk, **Pánovics János**, Gunda Lénárd: An evaluating tool for programming contests, *Teaching Mathematics and Computer Science* (2005) **3** (1), pp. 103–119.

Ref. number: Zentralblatt MathEduc Database ME 2005f.02643. (http://www.zentralblatt-math.org/matheduc/)

Book in the Subject of the Dissertation

6. Juhász István, Kósa Márk, **Pánovics János**: *C példatár* (in Hungarian), Panem, Budapest, 2005.

Conference Proceedings

7. Kádek Tamás, **Pánovics János**: *Általános állapottér modell* (in Hungarian), 23rd International Conference on Computers and Education, 2013, Sibiu, Romania, pp. 214–218.

8. Kósa Márk, **Pánovics János**: *Megoldáskereső algoritmusok programozása funkcionális megközelítésben* (in Hungarian), 22nd International Conference on Computers and Education, 2012, Alba Iulia, Romania, pp. 294–299.

9. Kósa Márk, **Pánovics János**: *Programming artificial intelligence search algorithms in a functional approach*, New Technologies in Science, Research and Education, XXV. DIDMATTECH 2012 International Scientific and Professional Conference, 2012, Komárno, Slovakia, pp. 144–151.

10. Kádek Tamás, Kósa Márk, **Pánovics János**: *A ProgCont szoftverrel támogatott programozó versenyek tapasztalatai* (in Hungarian), New Technologies in Science, Research and Education, XXV. DIDMATTECH 2012 International Scientific and Professional Conference, 2012, Komárno, Slovakia, pp. 152–157.

11. Kádek Tamás, Kósa Márk, **Pánovics János**: *Programozó versenyek támogatása webes alkalmazással* (in Hungarian), 21st International Conference on Computers and Education, 2011, Cluj-Napoca, Romania, pp. 184–187.

12. Kósa Márk, **Pánovics János**: *Kétszemélyes játékok optimalizálásának lehetőségei mobil eszközökre* (in Hungarian), Conference on Informatics in Higher Education 2011, Debrecen, pp. 200–206.

13. Kósa Márk, **Pánovics János**, Tőzsér Tamás: *Az amőba játék optimalizálásának lehetőségei mobil eszközökre* (in Hungarian), 20th International Conference on Computers and Education, 2010, Satu Mare, Romania, pp. 150–154.

14. Kósa Márk, **Pánovics János**: *Object-oriented approach of search algorithms for two-player games*, 8th International Conference on Applied Informatics (ICAI 2010), Eger, Vol. II., pp. 29–34.

15. Kósa Márk, **Pánovics János**: *Kétszemélyes játékok lépésajánló algoritmusai objektumorientált megközelítésben* (in Hungarian), 19th International Conference on Computers and Education, 2009, Târgu Mureş, Romania, pp. 263–266.

16. Kósa Márk, **Pánovics János**: *Szoftverfejlesztés a Qt keretrendszer használatával* (in Hungarian), 18th International Conference on Computers and Education, 2008, Şumuleu Ciuc, Romania, pp. 179–184.

17. Kósa Márk, **Pánovics János**: *Keresőalgoritmusok objektumorientált megközelítése a Mesterséges intelligencia tárgy bevezető kurzusán* (in Hungarian), 17th International Conference on Computers and Education, 2007, Oradea, Romania, pp. 94–97.

18. Kósa Márk, Nagy Benedek, **Pánovics János**: *Megoldáskereső algoritmusok hatékonyságának vizsgálata az állapottér-reprezentációk függvényében* (in Hungarian), 16th International Conference on Computers and Education, 2006, Sovata, Romania, pp. 76–81.

19. Kósa Márk, **Pánovics János**, Gunda Lénárd: *An evaluating tool for programming contests*, 6th International Conference on Applied Informatics (ICAI 2004), Eger, Vol. I., pp. 163–172.

20. Várterész Magda, Nagy Benedek, Kósa Márk, **Pánovics János**: *A Mesterséges intelligencia tárgy bevezető kurzusának gyakorlatai a Debreceni Egyetemen* (in Hungarian), Conference on Informatics in Higher Education 2002, Debrecen, pp. 1103–1109.

Conference Talks in the Subject of the Dissertation

21. Kádek Tamás, **Pánovics János**: *Általános állapottér modell* (in Hungarian), 23rd International Conference on Computers and Education, 2013, Sibiu, Romania, October 10–13, 2013.

22. **Pánovics János**: *Projektközpontú szemlélet az informatikaoktatásban* (in Hungarian), Sapientia MatInfo Conference, Târgu Mureş, Romania, May 25–26, 2013.

23. Kósa Márk, **Pánovics János**: *Megoldáskereső algoritmusok programozása funkcionális megközelítésben* (in Hungarian), 22nd International Conference on Computers and Education, 2012, Alba Iulia, Romania, October 11–14, 2012.

24. Kósa Márk, **Pánovics János**: *Programming artificial intelligence search algorithms in a functional approach*, XXV. DIDMATTECH 2012 International Scientific and Professional Conference, Komárno, Slovakia, September 10–13, 2012.

25. Kádek Tamás, Kósa Márk, **Pánovics János**: *Our experiences on the programming contests supported by the ProgCont software*, XXV. DIDMATTECH 2012 International Scientific and Professional Conference, Komárno, Slovakia, September 10–13, 2012.

26. Kádek Tamás, Kósa Márk, **Pánovics János**: *Programozó versenyek támogatása webes alkalmazással* (in Hungarian), 21st International Conference on Computers and Education, 2011, Cluj-Napoca, Romania, October 6–9, 2011.

27. Kósa Márk, **Pánovics János**: *Object-oriented approach of search algorithms for two-player games*, 8th International Conference on Applied Informatics, Eger, January 27–30, 2010.

28. Kósa Márk, **Pánovics János**: *Kétszemélyes játékok lépésajánló algoritmusai objektumorientált megközelítésben* (in Hungarian), 19nd International Conference on Computers and Education, 2009, Târgu Mureş, Romania, October 8–11, 2009.

29. Kósa Márk, **Pánovics János**: *Keresőalgoritmusok objektumtumorientált implementációja a Mesterséges intelligencia tárgy gyakorlati kurzusain* (in Hungarian), Conference on Informatics in Higher Education 2008, Debrecen, August 27–29, 2008.

30. Kósa Márk, **Pánovics János**: *Keresőalgoritmusok objektumorientált megközelítése a Mesterséges intelligencia tárgy bevezető kurzusán* (in Hungarian), 17th International Conference on Computers and Education, 2007, Oradea, Romania, October 11–14, 2007.

31. Kósa Márk, **Pánovics János**: *Search algorithms at ACM contests*, 7th International Conference on Applied Informatics, Eger, January 28–31, 2007.

32. Kósa Márk, Nagy Benedek, **Pánovics János**: *Performance analysis of search algorithms depending on the state space representation*, 6th Joint Conference on Mathematics and Computer Science, Pécs, July 12–15, 2006.

33. Kósa Márk, Nagy Benedek, **Pánovics János**: *Megoldáskereső algoritmusok hatékonyságának vizsgálata az állapottér-reprezentációk függvényében* (in Hungarian), 16th International Conference on Computers and Education, 2006, Sovata, Romania, May 25–28, 2006.

34. Kósa Márk, **Pánovics János**, Gunda Lénárd: *PCRM: kiértékelő szoftver programozói versenyekhez* (in Hungarian), Conference on Informatics in Higher Education 2005, Debrecen, August 24–26, 2005.

35. Kósa Márk, **Pánovics János**, Gunda Lénárd: *An evaluating tool for programming contests*, 6th International Conference on Applied Informatics, Eger, January 27–31, 2004.

36. Várterész Magda, Nagy Benedek, Kósa Márk, **Pánovics János**: *A Mesterséges intelligencia tárgy bevezető kurzusának gyakorlatai a Debreceni Egyetemen* (in Hungarian), Conference on Informatics in Higher Education 2002, Debrecen, August 28–30, 2002.

Further Conference Talks

37. Kósa Márk, **Pánovics János**: *Kétszemélyes játékok optimalizálásának lehetőségei mobil eszközökre* (in Hungarian), Conference on Informatics in Higher Education 2011, Debrecen, August 24–26, 2011.

38. Kósa Márk, **Pánovics János**, Tőzsér Tamás: *Az amőba játék optimalizálásának lehetőségei mobil eszközökre* (in Hungarian), 20th International Conference on Computers and Education, 2010, Satu Mare, Romania, October 7–10, 2010.

39. Kósa Márk, **Pánovics János**: *Szoftverfejlesztés a Qt keretrendszer használatával* (in Hungarian), 18th International Conference on Computers and Education, 2008, Şumuleu Ciuc, Romania, October 9–12, 2008.

40. Kósa Márk, **Pánovics János**, Heinrich du Toit, Nyakóné Juhász Katalin, John Pilcher: *Felhasználói felület fejlesztése az EPICS rendszerhez* (in Hungarian), Conference on Informatics in Higher Education 2008, Debrecen, August 27–29, 2008.

41. Nyakóné Juhász Katalin, Kósa Márk, **Pánovics János**, Lajkó Károly: *Öt éves múlt a TMCS* (in Hungarian), Conference on Informatics in Higher Education 2008, Debrecen, August 27–29, 2008.

Further Publications

42. Kósa Márk, **Pánovics János**: *Programming Contest Result Manager*, Technical reports, 2009/9, Preprints No. 369, Faculty of Informatics, University of Debrecen.

43. Kósa Márk, **Pánovics János**: *Példatár a Programozás 1 tárgyhoz* (in Hungarian), e-learning material, MobiDiák project, 2005.

Software Products

44. Kádek Tamás, Kósa Márk, **Pánovics János**: *ProgCont*, 2011 (7585 lines of Java code, 2724 lines of JSP code, 2499 lines of SQL code).

45. Gunda Lénárd, Kósa Márk, **Pánovics János**: *Programming Contest Result Manager*, 2003 (10588 lines of C++ code).