Olympiads in Informatics, 2016, Vol. 10, 125–159

DOI: 10.15388/ioi.2016.09

Programming in Slovak Primary Schools

Martina KABÁTOVÁ[1], Ivan KALAŠ[1,2], Monika TOMCSÁNYIOVÁ[1]
[1]*Department of Informatics Education, Comenius University*
Mlynska dolina, 842 48 Bratislava, Slovak Republic
[2]*UCL Knowledge Lab, Institute of Education*
23-29 Emerald Street, WC1N 3QS London
e-mail: martina.kabatova@gmail.com, {kalas; tomcsanyiova}@fmph.uniba.sk

Abstract. In our paper, we want to present the conception of elementary programming in primary Informatics education in Slovakia and the process of its integration into ordinary classrooms. First, we will familiarize the reader with the tradition of so called 'Informatics education' in Slovakia and with the various stages of the process of its integration. We will formulate the learning objectives of the elementary informatics as a school subject in Slovakia (referring to Blaho and Salanci, 2011) and give reasons why we believe that it offers an important opportunity for developing informatics knowledge, computational thinking and problem solving skills. We will primarily focus on the presentation of our arguments why we consider programming (in the form rigorously respecting the age of the primary pupils) to be appropriate and productive constituent of learning already for this age group. Several recent research findings, presented by Ackermann (2012) and others support our position here. In the next chapter, we will present in detail the conception of elementary programming and how it is implemented in the continuing professional development (CPD) of primary teachers in Slovakia. We will examine which programming environments are being used, what kind of pedagogies and which specific learning objectives our teachers apply. We will list programming concepts and identify corresponding cognitive operations, which we find appropriate for primary pupils. Then we will present and analyse the CPD of our in-service teachers (and the position of programming in this process) which we have recently implemented in Slovakia. Another important element of our CPD strategy is the well-known Bebras contest (in Slovakia it is called 'iBobor' or 'Informatics Beaver'). In the next chapter of our paper, we will apply qualitative educational inquiry methods to examine how our conception of elementary programming has really penetrated into primary classes in Slovakia. We are also interested in how it is being received by the teachers and pupils. Through interviews with the teachers we will identify different aspects of the whole process and main risk factors, which may complicate or hinder the implementation. In the final chapter, we will study the tendency to develop informatics and programming at the primary level in the context of various research projects presented in the academic research literature. We will compare various key findings of other research projects with our own experience.

Keywords: educational programming, primary education, computing, computational thinking.

1. Introduction

In recent years many educators, education policy makers and scientists call for integration of what they call computational thinking into primary education of all pupils. In influential documents like (The Royal Society, 2012) a need to distinguish computational thinking (or computer science, informatics or computing education) from the ICT education is declared. However, reasons for such reflections differ – computer scientists and industry leaders feel that not enough young people (and only very few women) choose to pursue a career in computer science and they believe that if pupils become familiar with some informatics concepts (within their primary education already) they will favour it later in their lives and careers. Others believe that computational thinking is equally important and key skill as literacy and mathematical thinking and they call for a redefinition of literacy and for integrating digital literacy development into primary education for the sake of educating a fully developed citizens to live in the digital world.

While ICT oriented education is included in most national curricula, many countries do not pay any special attention to including other (and in our opinion more interesting and more important) informatics concepts. However, we can observe important new step in the UK by establishing computing as a compulsory subject in every school year since September 2014.

The situation in Slovakia is different from most of the countries. Informatics as a separate mandatory subject was established many years ago and for many years we have been systematically preparing teachers for teaching it. The authors of our national curriculum took a great care to include topics from core informatics along with learning of basic ICT skills.

Since Comenius University plays a key role in building National Informatics Curriculum (2011, 2015) we have a lot of experience with integrating educational informatics both into schools (primary, lower secondary and upper secondary) and into teachers' professional development (PD). Some activities with digital technology were recently nation-wide integrated into early childhood education (or children 3 to 6) as well: see e.g. (Pekarova, 2008) and (Kalaš, 2010).

The whole conception of informatics is a broad and interesting topic to study, but for the purpose of this paper we fully focus only on one of its topics, namely, on **programming at primary level in Slovakia**. We will discuss its conception and the process of its integration into ordinary classrooms. We will present and explain:

(a) The reasons and short history of implementing primary informatics as a modern core subject taught in Slovak primary schools.
(b) Why we consider appropriate to have separate school subject focused on ICT and informatics, while we also support integration of ICT across curriculum.
(c) What role primary programming plays in our conception of informatics education and what forms, methods and pedagogies we consider appropriate in this context.
(d) How we proceed with the implementation of these objectives through in-service teacher development.

(e) What problems we have encountered, how this process actually evolves in our schools, how the teachers read our objectives, which objectives and corresponding skills they have mastered, how they interpret them and finally – which factors of the implementation we and the teachers perceive as risky or unfulfilled.

Brief History of Informatics at Slovak Schools

First we will familiarize the reader with the tradition of informatics education in Slovakia and various stages of the process of its implementation. We will briefly describe how the informatics education was established – from the period of experimental education at upper secondary schools in late 60s and early 70s, to the current stage of informatics as mandatory school subject for students from grade 2 (i.e. 7 to 8 year olds) up to the mid-upper secondary stage (i.e. 16 to 17 year olds).

In early 70s some of the vocational technical schools began to prepare students for operating industrial machinery via computers. In these schools some students learned basics of programming in Fortran and Cobol. Students first designed their programs using flowcharts, then they prepared corresponding punch cards which were then taken to the computer lab. Students never saw the computer themselves since it was usually located in a different institution and it took up several rooms. In the late 70s some schools built their own computer labs. In some industrial towns (where most of the vocational technical schools were located) special central computer labs were established. At that time new study programmes were launched at universities called informatics (at the beginning called cybernetics).

In 80s most of the upper secondary schools opened special classes focused on informatics. However, appropriately qualified teachers were absent. In school year 1982/83 Faculty of Mathematics and Physics of Comenius University opened a new study programme focused on upper secondary informatics teachers' pre-service education. Those students had access to the university computer EC1010 where they could write and run programs in Pascal. Soon after that several universities began to build computer labs equipped with 8-bit computers (e.g., PMD-85, Didaktik Alfa, PP-01), often using a version of Basic as a programming language

Many activities designed to attract young people to informatics emerged – in 1985 a P category of International Mathematical Olympiad started (later transformed into a stand-alone International Olympiad in Informatics). A series of articles on programming in environments like Karel and Logo were issued in the Zenit magazine targeted at secondary school students. Since 1986 a school subject "Informatics and computers" became part of the National Curriculum. Special classes focused on algorithms and programming were established at several grammar schools and vocational technical schools.

In early 90s most of the upper secondary schools taught informatics. A special computer lab with several PCs was usually dedicated to this subject, mostly taught by specialized teachers. The educators inspired by success at upper secondary school developed an experimental informatics curriculum also for lower secondary schools. For example,

Kosicka Str. Primary School in Bratislava opened a class focused on programming. Pupils aged 11 to 15 wrote game-like programs in a visual programming environment called Comenius Logo (Blaho *et al.*, 1995) and (Tomcsanyiova and Tomcsanyi, 1997). In the second half of 90s computers become more affordable and many businesses and households acquired them. In order to train students to use computers effectively, informatics in schools changed its orientation and became more user oriented – students learned how to create electronic documents (in T602, a text editor of that time), use spreadsheet editor, send and receive e-mails, navigate files and operating systems etc. In late 90s the National Curriculum was revised to incorporate five main topics – Information around us; Communication through ICT; Problem solving and algorithmic thinking; Principles of ICT; and Information society.

In 2008 a new National Curriculum for primary and secondary schools prescribed to teach informatics as a mandatory core subject from year 2 (7 to 8 year olds) to mid-upper secondary stage (16 to 17 year olds). At all school years five main topics of informatics remain the same and they cover basic digital literacy, ICT user skills, programming and core concepts of informatics, hardware and other digital technology related concepts, digital safety and other information society related concepts. At different school years the topics are taught differently – first and foremost respecting the pupils, their age and developmental stage.

There is an intense initiative in Slovakia to integrate digital technology into early years (pre-primary) education as well. Through an EU funded project teachers in early years education centres (kindergartens) are being educated to use digital technology appropriately with their children. Programmable toy Bee-Bot have been introduced, see (Pekarova, 2008) and (Kalaš, 2010).

According to our anecdotal information, programming at upper secondary level is mostly done in Delphi or Lazarus environments, with more and more schools gradually switching to Python. At lower secondary schools, Imagine Logo and Scratch are popular programming languages. At primary schools most widespread environments are Thomas the Clown, EasyLogo and several other microworlds that have been created in our department (we will present them in chapter 3).

2. Elementary Informatics, Computational Thinking and Programming

In accordance with the recent report of Informatics Europe and ACM Europe (2013) we will use the term **informatics** when we are speaking about the broad scientific field behind the digital technology. For us "informatics" is also an umbrella term that includes computing, ICT, and digital literacy – basically all concepts that have anything to do with digital technology, information or theory behind them.

An effort to distinguish various fields within school informatics is apparent in the Royal Society report (2012). However, we use these terms in a slightly different way from definitions provided there. By the term **ICT** we understand a set of user oriented skills (e.g., using a text editor, spreadsheet editor, creating graphics, animation, working with sounds …). **Digital literacy** in our context is understood as a set of basic skills that

everyone should acquire during their education in order to use digital technology (not only computers but all digital devices) effectively, safely and meaningfully to solve their everyday problems and tasks.

To understand what exactly is covered by our informatics (as a school subject) let's have a closer look at the five main topics in next chapter of our paper.

2.1. *Informatics as a School Subject in Slovakia*

Slovak National Curriculum (2011) codifies the following core school subjects rooted in informatics science (however, since 2015 both subjects are unified as Informatika):

- "**Informaticka vychova**", or Elementary Informatics in English, for school years 2 to 4 (pupils aged 7 to 10), while whole primary education consists of year 1 to 4 (i.e. pupils aged 6 to 10).
- "**Informatika**", which translates as Informatics, for school years 5 to 11 (pupils aged 10 to 17) at so called lower secondary and upper secondary schools.

For each of these subjects there is about 1 lesson per week, usually in a computer lab. Besides these dedicated subjects many ICT (and some informatics) elements are integrated across subjects as well, but that aspect will not be discussed in this paper.

Informatics as a core school subject is designed for every pupil regardless of their gender, future career or highest level of education they will reach. Great emphasis is on the age appropriateness – the content and form should always respect developmental stage of the pupils.

In all school years the five main topics of informatics remain the same, their content is always designed to fit the specific age group. National curriculum of primary informatics is presented in detail in Blaho and Salanci (2011). At primary schools the five topics cover:

- I**nformation around us** is the most comprehensive topic that includes working with text, graphics and multimedia. At primary school, pupils explore data structures – simple tables, graphs, dictionaries and mind maps.
- **Communication via ICT** – pupils work with websites relevant to their interests; they learn to use a web browser, e-mail client and chat.
- **Methods, problem solving and algorithmic thinking** – pupils learn to solve various problems and write down solutions (using words, icons or specific commands), they learn to control an agent directly and later by planning commands in advance. They learn to understand the causal connection between the program and behaviour of the agent. In this paper we will focus solely on this part of school informatics – and specifically on the elementary programming.
- **Principles of ICT** topic deals with hardware parts of the computer (keyboard, mouse, display) and external devices. Pupils also learn to work with folders and files.
- **Information society** – pupils learn about risks involved in using digital technology, about privacy and about the impact of information technology on the society.

We believe that these five topics cover the same concepts as three topics suggested in the Royal Society report (2012): digital literacy, information technology and computer science – which resulted into introducing a new compulsory subject computing in the UK since 2014.

In some other countries there is a strong initiative to include computational thinking and informatics concepts into all school subjects, see (Barr and Stephenson, 2011), instead of creating a separate dedicated subject. However, Slovak tradition of "informatics" as a school subject is a long one (including corresponding pre-service and in-service teacher development) and we believe that informatics is a distinct and important scientific field that should have a similar position in the education as mathematics or physics. Another contributing fact is that it seems to be unreasonable to demand from all the teachers to learn informatics concepts or how to incorporate computational thinking into their respective subjects – according to our experience they already struggle with integrating basic ICT elements into their teaching.

2.2. *Programming as a Component of School Informatics*

The core topic in the National Curriculum (2011, 2015) of school informatics that is most interconnected with informatics as a science is named Methods, problem solving, and algorithmic thinking. In it we expect pupils to learn how to solve various types of problems, externally represent a solution, and use such representation as an object to think with about the problem. Carefully chosen problems and well thought out pedagogy can lead directly to computational thinking development and even rather deep into elementary programming.

The term **computational thinking** was introduced and later developed by Wing, who understands it as

> *"a thought process involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent"* (Wing, 2011).

Recently, an interesting study by Selby (2013) refines the definition of computational thinking as...

> *"a focused approach to problem solving, incorporating thought process that utilizes abstraction, decomposition, algorithms, evaluation, and generalization."*

Wing and several other authors call for incorporating computational thinking into formative education of children (Wing 2008), (Lu and Fletcher, 2009), (Lee *et al.*, 2011) and (Hu, 2011). In some countries the focus is still on implementing informatics education only into secondary school, see e.g. (Hubwieser, 2012) and (Settle *et al.*, 2012). Our main interest lies in developing computation thinking "from the bottom" – i.e. form preschool and primary education. However, it is difficult to choose appropriate form

and content when it comes to this target group. According to Piaget's theory of cognitive development (1993), in accordance with Hu (2011) and also according to our own experience, most children reach the ability to work with abstractions only around their tenth year and some of them even later. It is agreed in the relevant literature that abstraction is the very essence of computational thinking. And yet we believe that supporting the development of computational thinking can productively start at the age of 5 or 6 by conducting well thought introductory steps leading to what we call elementary informatics.

Our first steps begin with direct manipulation with objects without the need to abstract or represent the process that is involved in their manipulation. While these activities may not resemble computational thinking at a first glance, we believe they are good preparation for development of higher order cognitive skills.

By **elementary programming** we understand activities in which pupils perform certain problem solving tasks of controlling an agent or planning its future behaviour – in a digital environment (programmable toy, microworld, programming environment...). We strongly believe that elementary programming is an excellent means for developing, implementing and verifying problem solving skills within the domain of computational thinking. If initiated at the primary stage of education, we also call it **primary programming**. An interesting study on connection of computational thinking and programming can be found in (Selby, 2012). There are many age-appropriate tools and environments that allow us to design meaningful and engaging elementary programming activities while respecting children's developmental stage. We believe we comply with the Blackwell's definition of programming (2002):

> *"Programming involves loss of direct manipulation as a result of abstraction over time, entities or situations. Interaction with abstractions is mediated by some representational notation."*

However, several problems arise if we want to define elementary programming activities. As we have already mentioned above – children in our target group have not yet developed their abstract thinking, and so abstracting over time itself is a problem. On the other hand, we believe that many valuable activities can be conducted before any kind of abstraction is involved. Moreover, these activities often have other features that are compatible with programming (e.g., some sort of representation is being used; planning future behaviour is expected etc.). We believe that learning to think computationally and to program one's solutions can be done gradually by doing specific activities that only slowly lead to a true abstraction, decomposition of problems and generalization of solutions.

We are aware that some authors consider programming at such an early age to be at least disputable, see (Lu and Fletcher, 2009), some regard programming as a significant form of computing but mathematical in its foundation, see (Hu, 2011). In this context we perceive elementary (or primary) programming as a tool for developing early computational thinking skills. We believe that carefully chosen tools, activities and pedagogies are an excellent way of integrating both – elementary programming and computation thinking – into primary education of all pupils. We – in accordance with Resnick (2012) – *"believe in Papert's dream of computational fluency for everyone",* that

> *"children should learn to program their own animations, games and simulations – and in the process learn important problem-solving skills and project-design strategies" and that it is necessary "to expand the conception of digital fluency to include designing and creating, not just browsing and interacting".*

However, we need to carefully build these skills gradually, always thinking about age-appropriateness.

There are several educators that promote programming as a productive and engaging activity even for very young children. They are looking for age-appropriate forms and study how do children approach various programming situations.

Mogardo *et al.*, (2006) deal with a pioneer experiment of Perlman who in 1970s designed a programming tool for preschool (and preliterate) children. The TORTIS system consisted of physical floor turtle that was controlled by logo-like commands depicted on plastic cards. Cards were placed into slots and after pushing a button they were executed by the floor turtle. Both the agent and the commands were tangible. Authors themselves admit that in the time of the described experiment there was only a little understanding of developmental psychology of a child and Perlman probably had not designed the tool in accordance with what we now would consider appropriate for such young children (the paper describes working with 3–5 year olds). However, some of her observations (analysed in 2006 by Mogardo *et al.*) are valuable even now – e.g., that children didn't manage to associate the screen commands with the movements of the turtle and even after adding the plastic cards (which were basically physical representations of screen commands for the turtle) children failed to understand that each card represents a movement of the turtle. We believe this problem is closely associated with a cognitive developmental stage of the children – at the age of 5 they definitely do not possess the ability to understand the connection between the plastic cards picturing commands and movements of the turtle on the floor. Our suggestion is to conduct different pre-programming activities that do not involve external representations (e.g. playing with Bee-Bots or even more trivial tools that involve "one command, one move" direct manipulations of the agent at a time) – and leave the programming activities involving abstractions and representations to later stages when pupils begin to develop the understanding of abstraction and external representations.

Ackermann deals with young children and their programming adventures in (2012) where she describes three aspects of programming as observed by the work with preschool children:

> *"1) making things do things (instruct them to follow and execute orders); 2) animating things (endow them with a mind of their own, teach them to look after themselves); 3) poking things (modulate how things act and interact by tweaking some parameters in their environment)."*

Ackermann admits that this is hardly a definition of programming per se and that the concept of programming is difficult, ever-changing and bearing many meanings to different people of different professions. However, she agrees that "*programming, at*

its core, is about giving instructions – or commands – to be executed by a machine". She presents several "*settings where youngsters are asked to give and execute orders, take over control*". For example – ambient programming is a new promising style that has a potential to attract many children, even those who in general do not incline to more traditional programming activities. Another take on ambient programming is described by Eisenberg (2009). On the other hand Ackermann refers to these activities as "*programming (in a weak sense)*" and she puts the word programming into quotation marks. In agreement with this approach we also distinguish our activities from *hard programming* and we will refer to them as *elementary programming* or *primary programming*. Another approach to programming, currently getting growing attention and becoming more widespread in all stages of education is physical computing and educational robotics programming, see e.g. (Przybylla, Romeike, 2014) or (Mayerova, Veselovska, 2016).

An interesting attempt at programming with primary school children is reported also by Gibson (2003). Probably the most successful initiative for programming for children is the Scratch community. Programming environment is being developed by researchers at the M.I.T. and it is continuously being improved and thoroughly studied, see Maloney *et al.* (2009) and Brennan *et al.* (2012).

3. Slovak Conception of Primary Programming

In this chapter we will present the conception of programming in Slovak primary education, based on current National Curriculum (2011, 2015) and materialised in the structure and content of the recent nation-wide professional development (PD) project for 700 primary teachers (see chapter 4). We will briefly characterise programming environments that have been used in the PD sessions and are currently being used in primary schools, what kind of pedagogies teachers apply and what are their learning objectives. We will analyse programming concepts and identify corresponding cognitive operations, which we consider appropriate for primary pupils.

In the Slovak approach to primary programming we can identify three domains with several sub-domains (with several overlaps and without any strictly predefined order of implementation, although with numerous dependencies in developing programming concepts and operations):

1) *Solving problems and handling solutions.*
2) *Controlling an agent:*
   - *Direct control of an agent.*
   - *Indirect control (building and handling future behaviours).*
   - *Some advanced concepts of primary programming (e.g. parameters, loops and procedures).*
3) *Tinkering with interactive environments:*
   - *Multiple agents and their properties.*
   - *Static scenarios.*
   - *Dynamic scenarios.*

3.1. *Solving Problems and Handling Solutions*

One of the main learning goals of primary informatics is to learn how to solve problems, and represent, evaluate, verify and reflect on their solutions. We consider concepts and practices in this domain to be exceptionally productive and developmentally appropriate. They can naturally contribute to all topics of primary informatics – including elementary programming.

In informatics education we focus especially on the procedure of problem solving which leads from the initial to the final state (the solution) while keeping given rules. Pupils probably do not perceive the procedure as the most important part of solving problems, but from the perspective of informatics education it is the core of the problem solving – the product (drawing the house, cannibals and missionaries transported to the opposite bank of the river, getting to the target square of the 'snake-and-ladder' game) is only a means of motivation. Therefore we choose problems that have interesting solving procedure (method or steps). We should always focus on the procedure of finding the solution and on its externalized representation (see Fig. 1). We should not neglect to verify if pupils are able to execute, communicate, analyse, evaluate and modify the discovered method of solution.

When designing lesson plans dealing with problem solving, it is important to choose both appropriate problems and learning activities to be conducted during the lesson. For some problems there exist supporting digital environments that enable pupils to solve them through direct interaction and visualization. This hands-on approach to solving problems supports experiments, iterative solutions, repeating solutions and trying out different solutions.

For example: the well-known puzzle about transporting the wolf, the goat and a cabbage across the river using one boat is an ideal problem for implementing via a software environment (Fig. 2 left). By clicking objects pupils experiment with transporting them.

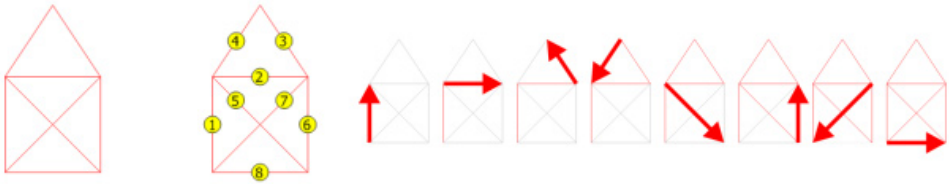

Fig 1. The first image is the required outcome – a one stroked drawing. The second and the third images (the third one being in fact a sequence of images) are possible notations of the procedure of how to solve it. Both solutions demonstrate how a solving procedure can be noted.

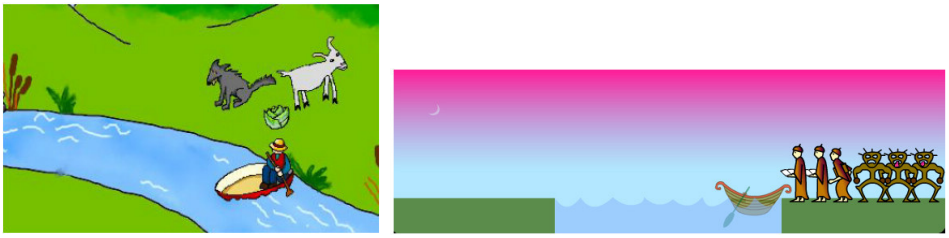

Fig. 2. On the left the Wolf, Goat and Cabbage puzzle environment. Right: screenshot from a similar puzzle with missionaries and cannibals, see `http://game-game.sk/18394/`.

Usually they solve the puzzle by trial-and-error method. Though, when they are asked how they have done it, they are motivated to reproduce and describe the solution and some of them are easily prompted also to put down the sequence of steps for transporting all items successfully.

If the environment is well designed it makes it easy to proceed from (1) **solving** the problem, through (2) **experimenting** with the solution procedure to the (3) **representing/recording** the procedure for future repeated solving of the same problem (maybe even without its immediate execution). These three steps in fact describe the advancement from solving problems to programming.

These are computational **cognitive operations** that are involved while solving problems and handling their solutions:

- Discuss and think about the core of a problem, about the relevant information provided by the problem assignment, about the conditions of solvability, about an appropriate procedure that will find a solution, about the difficulty level of the given problem, to look for similar problems that will help us to solve the problem.
- Use different strategies for finding the solution – like drawing a diagram, listing all combinations, guess-and-confirm, divide problem into smaller parts, find a similar problem, find a repeating pattern, look for the solution form the end etc., see Polya (1957).
- Explain the solution to someone else, to teach a friend how to solve it (verbally, by non-verbal means, using a specific language).
- Learn from someone else how to solve the problem (using verbal or non-verbal communication).
- Write down the solution (by a picture, or series of pictures, using icons, text, video or audio).
- Reason about the language and the form of notation of the solution in order to make it eligible for others.
- Execute the solution and verify its validity, correct wrong steps of the solution.
- Review certain properties of the solution (its eligibility, length, 'price'…), assess and compare it with several different solutions.
- Look for different solutions of the same problem.
- Reason about the non-existence of the solution.

3.1.1. *Activities and Examples*

An interactive microworld inspired by a task from the Bebras contest (see e.g. Dagienė and Stupurienė, 2016) enables pupils to experiment with sorting a group of children according to their heights (Fig. 3). It is possible to switch two children by clicking on the first one then the second one. This microworld also records the steps of the solution into a text file. A teacher or a researcher can use it to find out what strategy pupils used – if they all solved the problem similarly or if they applied different strategies – systematic or more random.

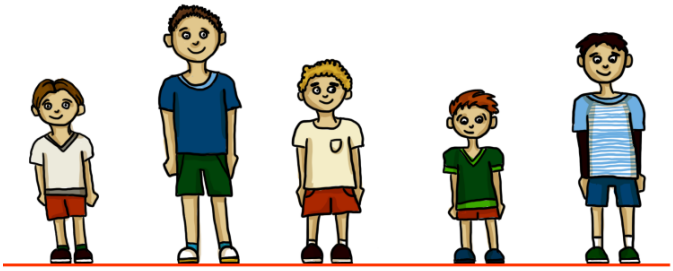

Fig. 3. Interactive application for a problem solving task.

3.1.2. *Pedagogy – Observations and Recommendations*

The specific organization of the lesson is up to the teacher – she is responsible for choosing the problems and selecting the activities according to the learning goals of the topic. Teacher can use many ready-to-use applications, microworlds and pre-made lesson plans. Many problems can be solved without the computer. Many tasks from the Bebras contest are suitable and they are available through the Bebras portal. The activities should be built around a direct manipulation with physical objects; or if they are implemented via some software application they should use appropriate pictures (dice, beads, building blocks, animals, persons …). It is crucial to motivate the pupils to actively think about the solution method and not to focus only on the end product – i.e. they should realize the difference between the procedure of drawing a house by one stroke and the resulting drawing where they can see no longer how it was done.

3.2. *Controlling an Agent*

Second domain of primary programming deals with two important and crucial concepts of pre-programming activities – direct and non-direct control of an agent. The first one is represented by a set of activities and suitable software applications that allow pupils to command an agent (a toy, another child, an on-screen character or animal etc.) to do something – mostly, to move to a given location. Each command is immediately executed and a result can be observed. Non-direct control of an agent gets pupils into real programming – they are asked to construct a sequence of commands in advance, which is then executed.

3.2.1. *Direct Control of an Agent*

Direct control of an agent can take place in a physical world where the teacher conducts an activity during which pupils give other pupils certain pre-defined commands (e.g. turn left, walk) to solve a given task (e.g. guide your friend from the desk to the door). Sometimes the commanded child can be replaced by a toy that is moved by hand according to the commands. There are also toys that can be controlled by a remote control, or digital toys with control buttons placed directly on them. Ambient programming

can also have similar characteristics, see (Eisenberg, 2009). In a software application a pupil controls a virtual agent. It is crucial for such microworlds to maintain age-appropriateness – using child-friendly graphics, presenting an engaging agent that a child can identify with or that can be perceived as a hero protagonist. In both cases (physical and virtual) the agent can execute only small set of well-defined basic commands – *make a step*, *turn left*, *turn right*, *play a sound*, *take an object*, *pick a colour*, *set a pen size*, *turn pen down* etc. In many software applications the virtual agent can be controlled directly but also by programming, i.e. without immediate execution of the command (see the next chapter 3.2.2.).

The most basic task for an agent is to move from one place to another. This task is often motivating enough and pupils are willing to carry it out and think about the sequence of commands to accomplish this goal. They gradually realise that:

- Only a limited set of commands is available to use in the solution (in a physical environment the teacher determines them, in a microworld they are usually set by the application itself).
- The current state of the agent is always represented by its visible attributes: rotation, position, pen colour etc.
- The execution of each command has a very concrete, specific and unambiguous effect on the agent and/or on the whole scene where it acts.

We choose the agents so that pupils are familiar with them and the activities they perform are more or less grounded in their reality (e.g. a bee flies to the flower, an ant moves objects) or at least actions of the agent should be believable (e.g. a turtle moves around and draws a line with its tail). Most agents therefore are animals, vehicles or human characters.

Many cognitive tasks listed in part 3.1. can be practised using activities mentioned in this chapter – by directly controlling the agent pupils can reason about the procedure of finding the solution, they can explain their solution to a friend, they can review specific properties of a given solution and it's correctness, or think about possible notation of the solution.

Activities and examples

Each of the software applications that will be presented in this chapter has its own specific features. They use different agents and different control interface, some of them record a sequence of commands. If the sequence of the steps can be recorded, we should consider its level of abstraction – the commands could be e.g. coloured pieces of paths (Thomas the Clown) or arrows that guide the agent (World of the Ant, Bee Tasks). Another significant difference in various microworlds and applications is whether the agent moves in a rectangular grid (Ice Cubes, Bee Tasks, World of the Ant, EasyLogo, Baltie), in a graph (Thomas the Clown) or with no visible constraints (Scratch). Rotation mode is closely related to the movement and the grid type – the agent can rotate either relatively or absolutely. Relative rotation means that the agent turns depends only on its previous heading; this is most common in open complex environments (Baltie, EasyLogo, Scratch). Absolute rotation is common in simpler applications where the agent moves in rectangular grid, usually only in four possible directions.

The agent in **Thomas the Clown** application is the clown on the bicycle. He moves in the graph-like network of roads. The child controls him by clicking the blue, yellow or red road piece in the right centre of the screen (see Fig. 4 left). The commands are executed immediately but the sequence is also recorded at the bottom of the screen. The task is to get Thomas from one place on the map to another.

In the **Ice Cubes** microworld the robot pushes the ice cubes (Fig. 4 right). It is controlled by the keyboard keys and the task is to move all ice cubes to their designated places. The sequence of moves is not recorded. Many similar microworlds are available on the web, though they are often perceived as games without educational dimension. Both applications check if the solution is correct.

The **World of the Ant** application features an Ant as the agent (see Fig. 5 left). The goal is to guide it through the maze to the door. Pink flower and blue star enable the Ant to walk through colourful walls. The Ant is controlled by the keyboard keys. The sequence is not recorded.

In the **Bee Tasks** the agent is a blue insect controlled by clicking the buttons with arrows (see Fig. 5 right). The goal is to guide it to the flower. The sequence of the commands is recorded on the bottom of the screen. Both microworlds verify whether the solution is correct.

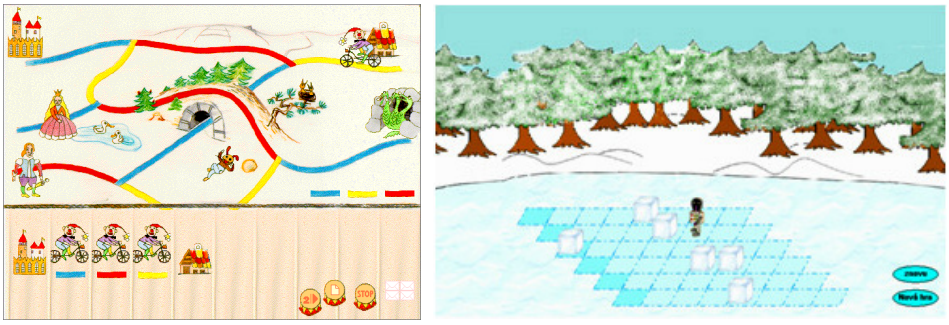

Fig. 4. On the left Thomas the Clown application, on the right an Ice Cubes microworld.

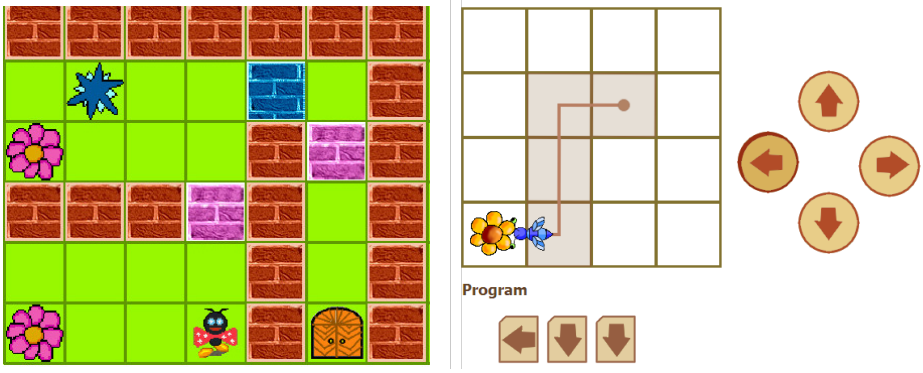

Fig. 5. On the left the World of the Ant application, on the right the Bee Tasks.

EasyLogo is open environment in which it is possible to change the appearance of the agent (in this case a frog). The agent is controlled by three buttons in the top right corner (Fig. 6 left). Forward arrow moves it one step forward on the grid – the agent moves along the grid lines, not from centre of the square to the next square. The left and right arrows turn the agent relatively to its current position. The goal is to guide the frog to the pond. This application does not check whether the solution is correct, nor does it record the sequence of commands in the direct mode.

Baltie is another open environment (Fig. 6 right). The grid where the agent (a sorcerer named Baltie) moves is not visible. Similarly to EasyLogo there are three buttons for the movement – first turns the agent relatively to the left, second moves Baltie one step forward and the third turns him relatively to the right. Baltie can conjure pictures – a child can choose pictures form the huge pre-prepared set. The picture will appear in front of Baltie and it is also possible to construct more complex images consisting of many smaller pictures. There is no in-built control of the correctness and the application does not record the sequence of commands in this mode.

Some of these applications allow creating and adding custom tasks for pupils – World of the Ant and EasyLogo. In World of the Ant we cannot choose a different agent or change the final goal of tasks, but we can design the maze and place object on different positions. EasyLogo is more opened – it allows to change the agent to any picture, set different backgrounds and completely rephrase the goal of the task (e.g. instead of guiding a frog to the pond we can ask pupils to move the frog along a square shape). We consider this an important feature – the teacher can design her own tasks which are better suited for the pupils and their skills, or match the motivation for the specific lesson. However, designing new tasks, creating custom pictures and related technical obstacles put a lot of demands on the teacher.

Open programming environments, such as **Baltie**, **Scratch** or **EasyLogo** can be used for the direct agent-controlling activities as well. However, a meaningful task has to be designed (or programmed) by the teacher first. There is no in-built solution checking and if the teacher needs such feature she has to virtually create the microworld to achieve this. A pre-made and partially programmed activity e.g. in Scratch can simulate desired

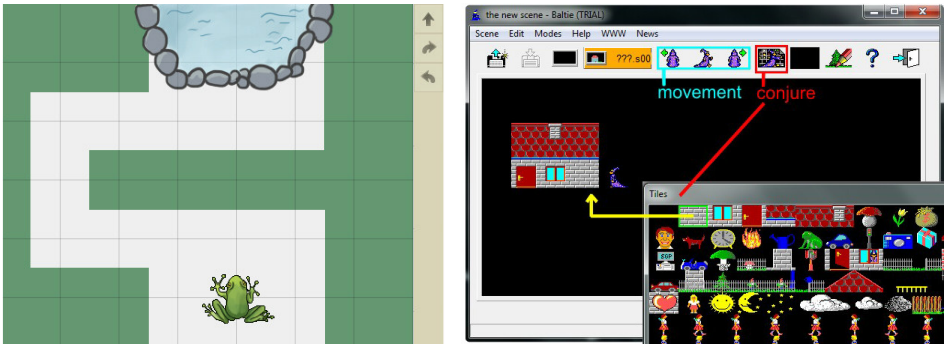

Fig. 6. On the left is EasyLogo, on the right is Baltie.

features of direct agent control. A path for the agent is in the following example a part of the backdrop (Fig. 7 right). The cat is controlled by keyboard arrows – this behaviour was programmed by the teacher in advance. Teacher also included a script that checks whether the cat is at the end of the path or whether it deviated from the path earlier. Similar assignments can be added to EasyLogo (Fig. 7 left), but it is not possible to add automatic solution checking.

Pedagogy – observations and recommendations

A teacher conducting these or similar activities must remember that the learning goal is to build basic understanding of controlling an agent by specific commands. The pupils should realize there is a causal connection between the commands and the behaviour of the agent. Since the control is direct and each command is immediately carried out, making this connection is possible for pupils before reaching formal operational stage of their cognitive development. In each application we presented the behaviour of the agent is visualized. This allows pupils to immediately see how they are progressing in the solution. We recommend using activities or environments that automatically check if the solution is correct. According to our experience, pupils are more motivated to solve problems if they have immediate feedback on their success. In our approach we always use direct control of an agent as an introductory activity to the very basics of elementary programming.

3.2.2. *Indirect Control of an Agent – Building and Handling Future Behaviours*

In the previous chapter we described activities in which the agent immediately carried out each command. Next step may naturally be focusing on planning the whole sequence of commands which will be executed only once it is complete. We call this approach an indirect control of an agent. Here again we can control either a physical agent (a classmate, a toy, or special programmable digital toy such as a Bee-Bot that is designed for that purpose) or a virtual one "living" in a software application on the screen. When working with physical agents pupils can write down the sequence of their commands on the paper, or draw it using pictures (an interesting activity by itself is to design the proper notation and discuss what 'proper' means in this context). E.g. programmable Bee-Bot

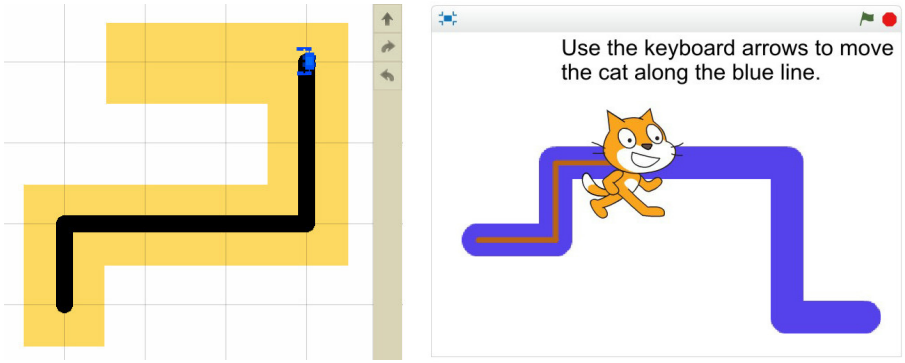

Fig. 7. Similar assignments in EasyLogo and Scratch.

does not display the sequence at all – it is entered by pressing the buttons atop the toy, but the child has to remember it or observe it when the toy moves according to the commands. When using software applications, notation is usually given by its designers – some applications use icons with arrows, icons combined with text or different kind of pictures (e.g. colour of the road in Thomas the Clown).

Indirect agent control assignments usually have the same goal as activities described in the previous part – to guide the agent from one place to another. Again it is possible to include also other actions e.g., picking and using items or avoiding obstacles. Since the sequence of commands is explicitly recorded – and thus visualized and editable – pupils can finish the sequence or add missing commands, or even correct the sequence, i.e. work with the representation. We can classify the cognitive operations according to what has to be done with the sequence of commands:

- Construct the sequence that guides the agent from its initial position to a final required position according to the assignment.
- Interpret a given sequence of commands (there are various ways of how to verify the interpretation, the most straightforward is when pupils move the agent according to the given sequence e.g., by clicking on the grid squares).
- Identify the final position of the agent after executing the commands.
- Identify the correct sequence among several sequences (more advanced version is to identify an incorrect sequence among several correct ones).
- Complete the sequence if the last step is missing, or two last steps are missing, or any step is missing.
- Identify and correct an incorrect command within the sequence.
- Find alternative solution, find a solution with specific properties (e.g. the path is the shortest possible, or on its path the agent will cross equal count of yellow and blue squares etc.).

Activities and examples

While most activities described above are suitable for implementation in a virtual microworld, it would probably be unreasonable to include all possible types of assignments into one environment, thus getting too complex or too much time consuming for primary pupils. Therefore several different applications are being used in our primary schools that focus on specific tasks or certain groups of tasks.

Indirect controlling of an agent in **Thomas the Clown** is implemented e.g. in the strawberry picking task (Fig. 8 left): the robot is waiting at the entrance to the garden, once a player completes a sequence of commands for moving and picking the strawberries, the robot will execute it. The goal is to pick all ripe red strawberries in the garden. Sequence is created by clicking the icons in the left part of the screen and it is recorded on the panel above the stage. When the sequence is being executed the active command is always highlighted. This microworld automatically generates different gardens of 2 by 3 grid squares.

In the **World of the Ant** (Fig. 8 right) we can also choose indirect control mode. At the bottom of the stage there is a set of commands – four arrows are for absolute rotation

and icon with legs represents moving one step in to the direction set previously. The Ant will carry out the sequence only after the child pushes the red button. The application is open to design many different mazes.

The **Bee Tasks** microworld was designed to offer several possible types of tasks we mentioned earlier. In its current version there are nine types – the first one was already described in part 3.2.1. as it is a direct control of the agent. The others are: interpreting a sequence of commands, constructing a sequence (the Bee has to get to the square with the flower, see Fig. 9 left), placing the flower on the square where the Bee will end up after completing the given sequence (again interpreting the sequence), adding a missing command (last one, last two, any in the middle), constructing a sequence (the Bee has to end in the square with the flower, but there are obstacles as well), and identifying a right sequence among several given (Fig. 9 right). A sequence is constructed by clicking on the icons with arrows – based on the same principle as in Thomas the Clown microworld.

Fig. 8. On the left Thomas the Clown, on the right World of the Ant.

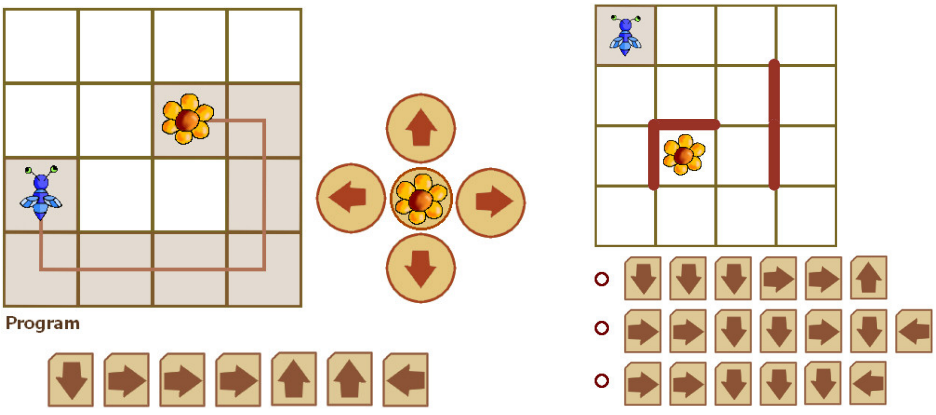

Fig. 9. Two different assignments in the Bee Tasks microworld.

EasyLogo offers a mode in which children plan and construct the sequence of commands (Fig. 10 left). However, this application behaves rather differently – it executes the commands immediately after they are dragged into the sequence – it doesn't wait for completing the sequence (commands are in the column at the right side of screen). In this sense the agent is directly controlled. However, there is a button "Run again" which re-executes the sequence after it is created.

If we want to use **Scratch** for this kind of activities, we first need to prepare a project – create a suitable backdrop, create sprite(s) and build the scripts for all functions, including script(s) to verify a solution (if we want to). An example below (Fig. 10 right) is an activity in which the goal is to build a sequence of commands for the Beatle to move from its green (start) square to another green (goal) square without even touching any other non-white square. The backdrop is a grid of white and coloured squares. In the Beatle's scripts area there are four blocks prepared for the pupils – already with their inputs properly set (move to the centre of a neighbouring square and turn left or right 90). Pupils will construct the whole solution (script) for that situation by duplicating and snapping the blocks into one script. This task has many variants of different levels of difficulty.

In both of these examples the task could be to fill in one or more missing commands into the incomplete sequence (solution). Some of the environments presented so far offer an option for the pupils or for the teacher to add their own tasks of the same kind. Scratch, World of the Ant and EasyLogo allow us to do so.

Similar tasks (where the goal is to guide the agent to a given goal) are used also in the **Bebras** contest. Since 2010 primary pupils can be involved in the contest in a special category specially designed for them. One task was inspired by Thomas the Clown (Fig. 11 left) – the farmer has to get to his cow, but on his way he needs to grab the bucket. All possible paths are depicted as a graph with edges of different colours. Pupils are prompted to select the track which meets the criteria.

Second example from the Bebras contest is quite difficult task that proved to be problematic as only 20 % of pupils solved it correctly, 17 % didn't answer at all. The story is: "A mouse is roaming in the maze, until it eventually reaches the cheese. Philip was observing the mouse and used small cards with arrows to record its movements. Unfortunately he dropped the cards and only two of them stayed at their places (see the

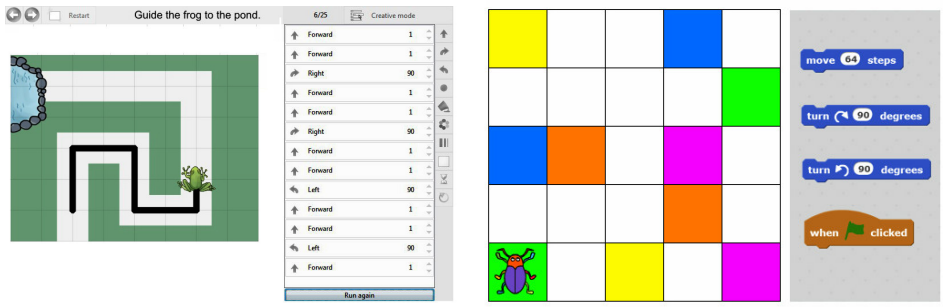

Fig. 10. EasyLogo and Scratch.

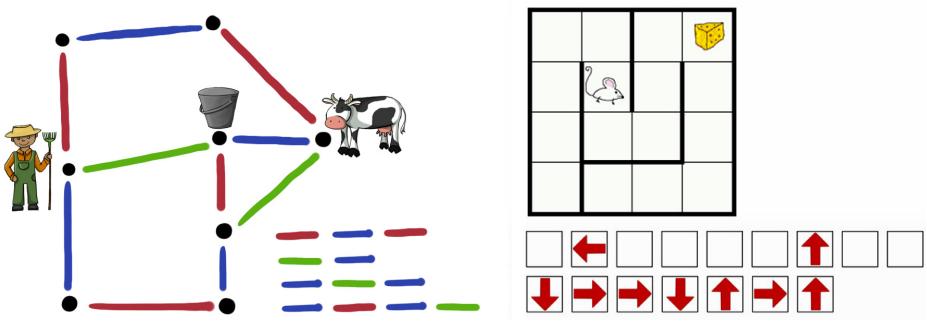

Fig. 11. Two Bebras tasks based on indirect control of an agent.

upper row of squares). Place the remaining cards and restore Philip's record." The task was interactive and pupils could drag the cards into empty slots using mouse.

The last described assignment illustrates that this kind of activity can be really difficult and can also be given to much older pupils. The variety of presented tasks show that guiding the agent is very rich context with many possible activities and a lot of potential. At the same time it is apparent that planning a sequence beforehand and executing it only after it's recorded is a programming-like activity that involves abstraction over time. Also the specific notation and execution of the sequence by some automatic machine-like agent is a feature of full-flagged programming activity. However, these tasks are still set in a concrete situations and their solutions do not require pupils to design universal solutions that involve this kind of abstraction.

Pedagogy – observations and recommendations

It proved to be crucial that the application itself verifies whether the solution is correct. If the microworld offers several tasks or several levels of difficulty, pupils should not be allowed to skip them freely. Most motivating environments have a game-like design presenting a bit more difficult task in each level. The designers of the microworld should always prepare a set of tasks to be solved by pupils. They should be ordered according to their cognitive demands, they should be motivating and engaging, prompting pupils to learn new concepts and challenging to engage more demanding (but still developmentally appropriate) cognitive operations. It is useful if the designer prepares several sets of tasks as they can be used for achieving different learning objectives, in different classes, for pupils at various stages of the learning process. Interesting option is to allow teachers to create their own tasks, however this approach has proven to be far too optimistic as only a small fraction of teachers are ready to do so.

3.2.3. *Classification of the Microworlds Used for Direct and Indirect Control of a Virtual Agent*

In part 3.2 we have presented several applications, microworlds and environments that are suitable for solving problems and learning computational thinking via programming-like activities. They are all suitable for primary school pupils as such or after certain preparatory steps. We summarize their features in Table 1.

Table 1
Features of microworlds

Movement commands	keyboard keys	World of the Ant, Ice Cubes
	icons with arrows	Bee Tasks, EasyLogo
	icons with agent image	Baltie
	icons with colours	Thomas the Clown
	cards with text	Scratch
Agent rotation style	without rotation	Thomas the Clown
	absolute rotation	World of the Ant, Ice Cubes, Bee Tasks
	relative rotation	EasyLogo, Baltie, Scratch
Grid type	graph	Thomas the Clown
	rectangles or squares	Thomas the Clown, World of the Ant, Ice Cubes, Bee Tasks, Baltie
	rectangular – lines	EasyLogo
	free movement (coordinates)	Scratch
Notation (in direct control mode)	without notation	World of the Ant, Ice Cubes, EasyLogo, Baltie, Scratch
	automatic notation	Thomas the Clown, Bee Tasks
Solution verification	no verification	EasyLogo, Baltie, Scratch
	automatic verification	Thomas the Clown, World of the Ant, Ice Cubes, Bee Tasks
Agent actions	only movement and/or rotation	Thomas the Clown, World of the Ant, Bee Tasks, EasyLogo, Baltie
	collecting objects	Thomas the Clown, World of the Ant
	moving objects	World of the Ant, Ice Cubes
	using objects	World of the Ant
	other	Baltie, Scratch
Goals	arrive at destination	Thomas the Clown, World of the Ant, Ice Cubes, Bee Tasks, EasyLogo
	other	EasyLogo, Baltie, Scratch
Pre-made activities	no in-built activities	Baltie, Scratch
	set of fixed inbuilt activities	Thomas the Clown, Ice Cubes, Bee Tasks
	set of activities provided, custom ones may be added	World of the Ant, EasyLogo

3.2.4. *Some Advanced Concepts of Elementary Programming*

In the previous two parts we focused on basic concepts that are in our opinion and according to our experience suitable and appropriate for all primary school pupils. Now we will present several others – more advance concepts – which still could fit into upper end of the primary programming, but probably not for the whole class and only with well experienced teacher.

Parameters

In some applications parameters in existing commands are rather intuitive and easy to use (e.g. in Scratch, Fig. 12 left). In this case there is no need to address this concept ex-

plicitly – children will understand immediately how to use them. Rather intuitive way to set the parameter is using a drop-down menu – in Scratch there are several – e.g. choosing a sound which will play by the "play sound" command. In this case pupils cannot make a mistake. Another child-friendly parameter is pen colour chosen from the palette (see EasyLogo displayed on Fig. 12 right). Parameters are present also in some modes of World of the Ant and in Baltie environment.

Loops

Several of described microworlds provide loops – Scratch, EasyLogo, Baltie. However, they are usually present in more complex open programming environments. Led by our department a small microworld focused on loop constructions was designed and developed (Fig. 13). In this application pupils control the Jumper who has to reach the door by jumping over platforms. The microworld is designed as a game – there are 24 levels in which the child solves more and more complex situations (it is also possible to design and add custom levels). Eventually the space for the commands becomes limited and pupils cannot solve the problem without using a repeat loop. This design proved to be highly motivating and pupils are deeply keen on completing the "game". On the other hand, one of the teachers using this microworld reported that only about a half of pupils aged 8 to 9 years were able to learn to use the loop themselves. Loops appear also in the LEGO WeDo programming language that is designed for primary schools. In this case,

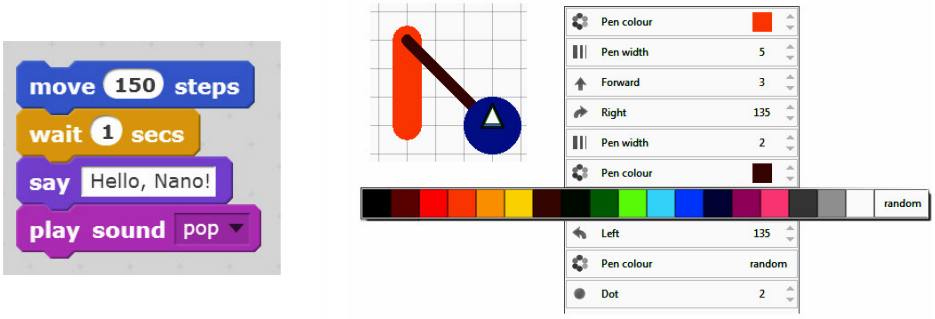

Fig. 12. Parameters used in Scratch and EasyLogo.

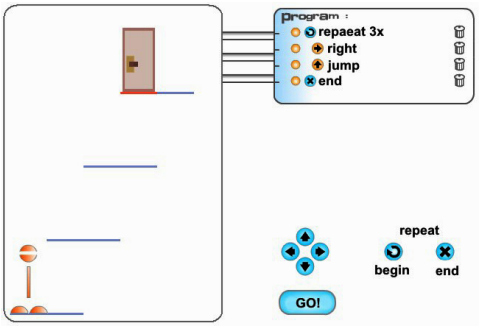

Fig. 13. A game-like microworld named Jumper is focused solely on loop constructions.

however, some commands implicitly contain repeated behaviour – turning on the motor means it will move until the program is stopped or some other action is assigned to it. As our research team reported pupils use the loops block rather easily and they intuitively understand their use in the programs they create for their robotic models.

Procedures

Some programming environments for primary pupils do not offer procedures (Scratch 1.4), others are designed to use them (EasyLogo, Scratch 2.0). According to our experience this concept is rather complex and is a good candidate to postpone to years 5 and 6. Here is an example of two procedures (Fig. 14 left) in EasyLogo. Pupils at first do not design them, as these loops are already prepared in the activity; she is prompted to use them in the program.

3.3. *Tinkering with Interactive Environments*

A programming environment named Living Pictures (influenced by Russian PervoLogo) has been specifically designed in our department to teach pupils some object-oriented concepts within elementary programming. In this environment pupils populate the virtual world (represented by a background) with moving objects – characters, animals, vehicles, plants or anything they choose from a pre-made set of pictures or draw them themselves. From the perspective of primary informatics pupils learn to control one or more objects, define their behaviours, clone them, set their properties and reactions to events.

Each object is at first depicted as a Logo turtle – the child should realise that this is in fact an abstract object that can take any form. Each object has different properties, its shape and position among them. Basic action of the object is its reaction to the onClick event (e.g. it can move few steps forward). Other events are triggered when the project is set to run and when objects collide, but we recommend to program these events later – with lower secondary school students, or with only some high achievers at the end of year 4. This programming environment is open – there are no pre-set activities. All

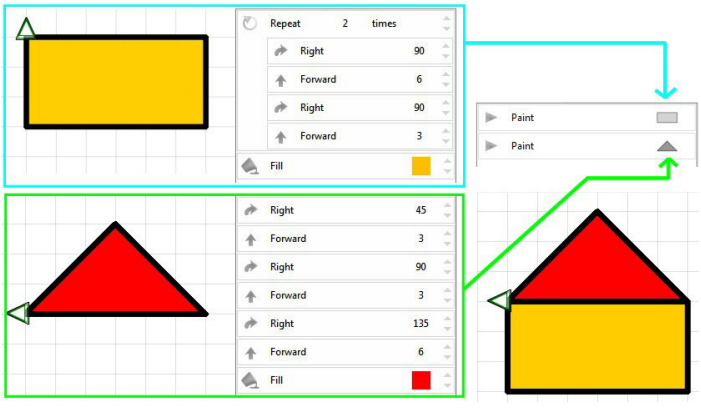

Fig. 14. EasyLogo procedures for drawing a yellow rectangle and red triangle are used to draw house.

assignments have to be designed and presented by the teacher. It is possible to add custom backgrounds and pictures. The teacher can adjust also the set of commands for the objects so that the pupils see only a limited sub set (Fig. 15). We consider this high level of customizability to be important especially if the application is designed for primary pupils. Setting up the user environment so that it is as simple as possible is crucial for the introductory lessons.

Another advantage of this application is that it is possible to export a project as an executable file (EXE). Thus pupils can be motivated to create moving pictures for their younger classmates (in accordance with Papert's principles of constructionist learning). Pupils can present the executable file to their friends or relatives.

In next parts we will illustrate several activities that can be done in this environment. We will focus on shape, position and rotation of the object, and we will use three events – onClick, onRunProject and onCollision. We believe that the outlined sequence of activities leads from designing a scene and setting the properties of objects to executing dynamic scenarios with multiple objects with different behaviour (which involves abstraction over time and over situation as well, see Blackwell, 2002).

3.3.1. *Multiple Agents and their Properties*

In parts 3.1 and 3.2 pupils controlled only a single agent. Using Living Pictures (or similar microworlds) it is possible to introduce multiple agents with different or identical properties. We prefer to tinker with properties that are visible – shapes, positions, and rotations. First, pupils should encounter objects with shapes that enable to see its rotation (character, animal …) later they will learn that for some shapes rotations are not observable (snowflake, sun). A good metaphor for describing such activity is a theatre – there are several actors on the stage, each of them has their own specific scenario and eventually they interact. This description helps with distinguishing the preparation phase (setting the properties, preparing sequences of commands) and the execution phase (the objects carry out their instructions).

Activities and examples

A good introductory activity is populating the world – pupils choose the background (green hills and sky) and place several objects on it, then change shapes of these objects

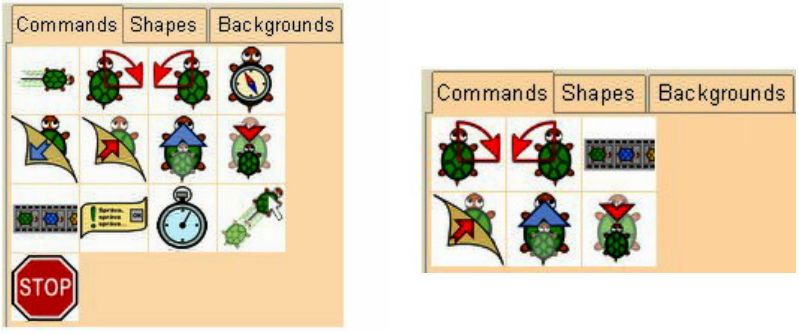

Fig. 15. Whole set of all commands (left) and limited set designed for a specific activity (right).

so they look like sun, clouds and trees. We can ask pupils to shrink or grow the objects according to their positions in the background so that an illusion of depth is created (Fig. 16 left). The features used here are: adding objects, changing their shapes, scaling down and scaling up and cloning objects. This activity can be done in many different settings – for example in the outer space (see Fig. 16 right). Rotation of the object can be also used in static scenarios (the astronaut is looking towards the aliens).

3.3.2. *Static Scenarios*

Clicking on or touching objects in the screen is nowadays the most intuitive way of interaction with the digital devices. This trend was set with the Windows interface and now is reinforced with touch screen technology. Objects in Living Pictures have a preset onClick even that is triggered if the object is clicked by a computer mouse. Pupils are already familiar with this event and assigning a reaction to the object when it is triggered is the next step.

Activities and examples

In Living Pictures each object has its own event window into which the commands for the object are dragged from the command palette. When designing static scenarios pupils will change the shape, size and rotation of the objects. The most straight forward activity is changing costumes. First the background is chosen, then objects are placed. For each object that is a piece of costume the pupils will set a behaviour – when it is clicked its shape will change to the next one form the chosen set of shapes.

As an example let us select a winter background with a snowman. Objects that will change their shapes with a mouse click are the hat, the broom, his face and buttons (Fig. 17 left). Similar projects are easily done in Scratch. The sprites have when clicked event and a single next costume block rotates a set of prepared shapes for the sprite. In our example (Fig. 17 right) three sprites can be clicked – clown's hat, eyes and mouth. Each object has the same and very simple script – switch the costume to the next one.

The greatest disadvantage of Living Pictures is that pupils can not immediately test the script and see what happens (they have to close the event window first and then run the project) – in Scratch it is possible. In Living Pictures it is also not possible to see the

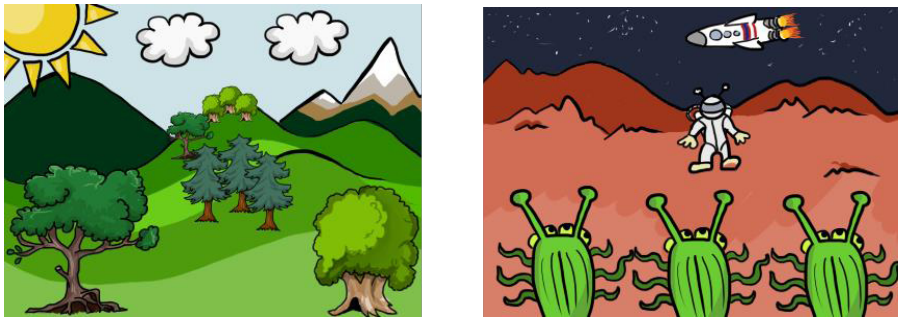

Fig. 16. Two static scenarios done in Living Pictures.

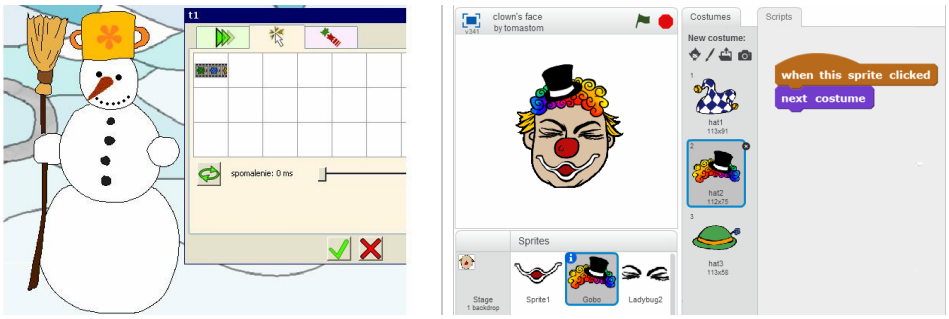

Fig. 17. Left: The "t1" window displays the onClick event with one command – change shape to the next one – which looks like a filmstrip. On the right similar project built in Scratch.

set of shapes for the object or which one comes next. In both environments we should encourage pupils to design the desired behaviour first for one object and only once it is tested and it works properly they clone it. This is done in accordance with object-oriented approach in programming where the programmer first designs the prototype object and its methods. Only after that is it reasonable to create inherited classes and objects with modified behaviour. Primary school pupils can learn to distinguish different objects with different properties and behaviour, or to tell what objects have in common. We believe this level of abstraction is appropriate for the primary pupils in the highest years (10–11 year olds).

3.3.3. *Dynamic Scenarios*

Dynamic scenarios in Living Pictures environment involve assigning a motion to the objects. Most common is setting an infinite loop for the motion, which is done by checking one of the options in the script (note that loop is not provided here as a programming structure). Throughout these activities pupils better understand the difference between preparation of the scene and running a project.

Activities and examples

In the Pond project pupils are prompted to set a background that will represent the pond. They place an object and change its shape so it looks like a fish. Then they set its direction – on our picture (Fig. 18 left) it will face right. In the event window they will command the fish to move forever forward. Default behaviour in Living Pictures is: if the fish is on the right edge of the screen it does not stop to move but it reappears on the left edge – objects do not bounce by default. This is a deliberate design choice that enables us to have an object which is forever moving to the right on a finite screen. After testing the fish's behaviour pupils clone it. Now they can change direction for some of them, or add some commands to onClick event (e.g. the fish disappears).

This is one possible set of activities in Living Pictures:

- Setting properties (shape, position, scale, direction) and cloning objects.
- One or more objects react to the onClick event.

- One object moves and reacts to the keyboard arrows; navigating this object is similar to direct agent control activities described in 3.2.
- Infinite movements of one or more objects; only one command is given to objects (usually move forward) and it is set to repeat forever.
- Infinite random movement of one or more objects on the scene, e.g. a butterfly is flying on a meadow, or Thomas the Clown is cycling on the plaza.
- Infinite (random) movement of one or more objects on the scene and their reaction to onClick event. This activity is on the edge of game design – pupils can prepare a scene where objects move randomly, when they are clicked they disappear. Aim of the game is to hit all the objects.
- Infinite (random) movement of one or more objects on the scene and their reaction to onCollision event. These kinds of activities are probably too complex for primary pupils, but if they are familiar with all previously listed concepts, they may be able to do them. An example: one object is a basket that reacts to the arrow keys, other objects in the scene are apples that are falling down (they move forever downwards), when the apple hits the basket it disappears. More complicated scenarios can be devised, but we believe there is too much abstraction involved and we do not consider this type of activity to be age-appropriate at Slovak primary level (consisting of only four years up to 10 years old pupils).

3.3.4. *Pedagogy – Observations and Recommendations*

We believe that tinkering objects, their properties and behaviours is an excellent opportunity for the primary pupils to learn the very basics of the object-oriented approach to programming. Activities in Scratch or Living Pictures are very intuitive. Pupils learn to change and set properties of objects, to distinguish the development phase form the running phase, to plan the future behaviour of objects, incorporate looping actions of objects and even begin to tinker with random values. It is crucial that these environments contain a large set of pre-made graphics and they should be opened to adding custom pictures. We believe that properly designed environment for tinkering with objects should:

- Allow to add object easily.
- Make changes in object properties (like shape, size or position) immediately visible.

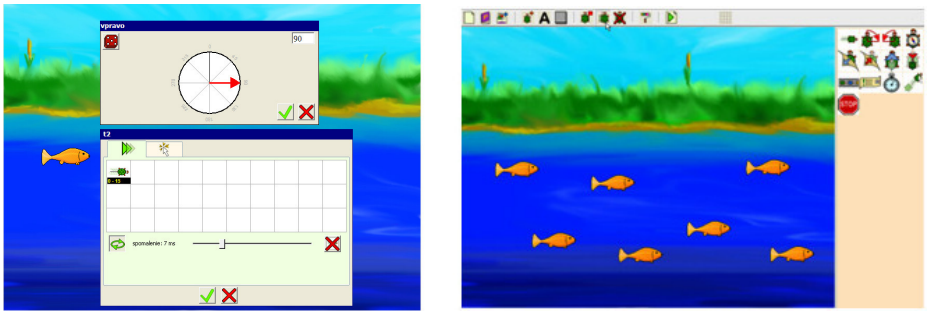

Fig. 18. The Pond project created in Living Pictures.

- Enable the object to react to at least three events: onClick, onRunProject, oOnCollision.
- Feature easy pupils-friendly manipulation with objects and their properties – by clicking and dragging.
- Enable cloning objects together with their behaviours.

3.4. *Implementation of Elementary Programming: Various Ways and Various Tools*

It is apparent from activities described in parts 3.1, 3.2 and 3.3 that core of primary programming can be learned using specialized software applications or microworlds that are (a) specifically designed for primary pupils and (b) designed to address the learning goals we have listed earlier. There are many similar applications being created around the world: Scratch (and its newer version Scratch 2.0), Scratch Jr., Microworlds JR, LEGO WeDo or Baltie. Several powerful microworlds have been developed in our department and made available for teachers and their pupils through various portals, websites, projects and PD sessions. Those include Thomas The Clown, World of the Ant, EasyLogo, Living Pictures, Jumper and Bee Tasks. Ice Cubes microworld and many similar ones originate from '*Infovekacik*' – an older Slovak on-line magazine for pupils created in cooperation with our department.

Another productive means to support implementation of primary programming into formal education for all children is the international contest Bebras. In Slovakia we initiated a special category for primary pupils and many contest tasks are deliberately designed to incorporate problem solving and elementary programming concepts. The national success and high number of contestants suggest that pupils and teachers are interested in this form of informatics.

4. Programming in Primary Teachers' Professional Development

All Slovak primary teachers have to get a master degree from a pedagogical faculty of one of our universities. They are not specialists – they teach pupils of years 1 to 4 (6 to 10 years old children) all subjects (sometimes excluding foreign languages and/or informatics). In 2008 a new compulsory school subject was introduced – primary informatics. However, pedagogical faculties have failed to update their study programs to include corresponding pre-service development for future teachers till today. Fortunately, a national project focused on professional development of in-service informatics teachers was launched in 2008 (till 2011) and authors of this paper were involved – together with the teams from five universities across the country – in developing its strategy and content and delivering it to 700 in-service teachers. The main goal of the project was to offer a modern, up-to-date, high quality education necessary for teaching this new subject at primary schools. Note that similar situation and PD strategy is being reported from the Czech Republic by Vanicek (2013).

Within the Slovak national project, 700 in-service primary teachers attended 18 study modules (each 6 hours long). Each module belonged to one of four tracks: *Digital literacy*; *Informatics*; *Didactics of primary informatics*; and *Modern school*. The Informatics track included six modules: *Computer and digital devices*; *Information around us 1–3*; and *Problem solving and basics of programming 1–2*. Teachers that graduated from the PD should be able to use digital technologies both in their classes and when preparing for them. They should perceive the elementary informatics as an important part of pupils' education and development, being able to meet the learning objectives of primary informatics as prescribed in the national curriculum. The study materials and whole lecturing process was designed to prepare the teachers for future development in digital technologies – for new microworlds and new operating systems, and also for new devices that would be used in the classrooms in near future. Authors of the study materials and lecturers took great care to introduce the teachers to a variety of software applications and appropriate teaching strategies. Teachers were learning how to evaluate appropriateness of software applications and microworlds and how to use them in the classroom.

From the perspective of this paper we find most relevant the modules dealing with **problem solving**, **elementary programming** and corresponding **didactical materials**. These areas had not been treated until then in any literature in our country (and hardly anywhere) and designing that content and delivering it to 700 in-service teachers was a real challenge and important innovative step towards new primary informatics. For the sake of the project, several new microworlds had been created, e.g. EasyLogo and Living Pictures, and participants used many other already existing microworlds and programming environments designed for primary pupils by experts in Slovakia.

One of the most successful new developments in the project was an idea and implementation of the Cards Tool (Tomcsanyi 2012). It is an authoring application that enables the teacher to design simple but vastly variable activities for any (primary) school subject. Participants of the project enthusiastically used the tool and created interesting activities that confirmed that primary teachers are creative and persistent and can use digital technology in their teaching. Since then, several thousand different activities created by the teachers themselves in the Cards Tool have been posted at Slovak portal `zborovna.sk`.

Although we lost touch with most of the participants when the project finished, we are interested in following how they manage to utilize new skills in their practice. Therefore we sporadically address a small sample of the participants and ask them to reflect about the project's longer term benefits. From that (mostly anecdotal) data we may formulate several interesting observations about the implementation of the *problem solving and programming activities* at primary level:

- Primary informatics lessons are usually run in a special computer lab, dedicated to primary key stage (older pupils usually use another computer lab).
- In each year group (2, 3 and 4) around 5 to 8 lessons are allocated to *problem solving and programming*. These are usually taught in a row, often towards the end of the school year.

- Teachers are using the study materials extensively and share them with other colleagues. They have altered their lesson plans to incorporate teaching methods applied in the project's PD sessions.
- Problem solving tasks (as presented earlier) are not highly popular among the teachers; many teachers simply skip them. They rarely realise that those tasks are not puzzles nor riddles, nor that their goal is not to find the solution by trial-and-error but systematically look for the solving method and reflect about the externally represented solution.
- Teachers enthusiastically use some microworlds that were created for the project – most of all The Jumper, World of the Ant and The Living Pictures while EasyLogo is less popular. Interestingly, each teacher has a strong preference for exactly one of the microworlds.
- Microworlds with the in-built sets of tasks of increasing difficulty and automatic verification of the solutions are used the most. Teachers often say they cannot provide immediate feedback for all pupils in the group and primary pupils are very keen on learning if they are progressing in the assignments. The sets should be designed to be solvable within one lesson (45 minutes). It should not be possible to skip the tasks in the set – only after the task is solved correctly the child can proceed to a harder one.
- Open programming environments are difficult to use and the teacher has to be better prepared for designing her own meaningful assignments and tasks within such environments (often it requires to attend extensive specialized training for the chosen environment). Our primary teachers probably have not reached that level of expertise yet.
- Most of the teachers are familiar with, visit and use the *Infovekacik website* – an on-line magazine for children with dozens of game-like microworlds. It would be probably useful to create a web portal with similar content and add lesson plans and recommended teaching methods. Teachers need good resources for their teaching that would inspire them to search for new suitable microworlds and software applications.
- Many teachers use The Cards Tool to design their own simple activities for other school subjects (mostly language and science, only rarely for primary informatics).
- All teachers are appreciative and see high value of the project's PD and of the new subject.

In conclusion, we believe that the national project and its PD programs were well designed and conducted. The participating teachers do incorporate learned skills and knowledge into their teaching. However, some of our plans proved to be too optimistic – most notably our inability to share with the primary teachers the importance and learning potential of the *problem solving tasks* (as described earlier in 3.1). Another failed expectation was to assume they would design their own sets of tasks for the pupils to support their informatics learning objectives. Teachers prefer to use the activities we prepared for them and their PD. Clearly it is vital to provide suitable series of activities with each microworld or digital toy/tool.

5. Discussion and Conclusion

Presented approach to primary programming has resulted from our previous experiences with teaching programming and developing programming interventions for all stages of schools, including university study programs for future teachers of informatics, for several decades. Our professional roots lie in Logo culture, into which our Comenius group has contributed by two internationally recognized versions of Logo: Comenius Logo and Imagine Logo. From that background we inherited our endeavour to respect the needs of students, together with other principles of Papert (1999) such as:

- *The Logo programming language is far from all there is to and in principle, we could imagine using a different language, but programming itself is a key element of this culture.*
- *So is the assumption that children can program at very young ages.*
- *And the assumption that children can program implies something much larger: in this culture we believe (correction: we know) that children of all ages and from all social backgrounds can do much more than they are believed capable of doing. Just give them the tools and the opportunity.*
- *Opportunity means more than just "access" to computers. It means an intellectual culture in which individual projects are encouraged and contact with powerful ideas is facilitated.*

We have also learned how important it is to integrate programming into pupils' learning experience only if they themselves see the meaning in doing so and perceive programming as a means to express themselves, to solve problems, *to make things happen*... In the case of primary pupils, such programming should most probably restrict to building simple future behaviours in certain notational system and solving tasks, which arise from handling such behaviours.

Although we consider elements of programming to be key constituent of informatics in primary education, we do not develop it as a means to attract more students to later Computer Science majors. We build it as a valued and legitimate core subject contributing to general education and complex development of every girl and every boy. Yet, we hope, that it may consequently play that role as well − the skills, knowledge, and attitudes, which pupils gain in elementary informatics may later help them build sound understanding of Computer Science principles.

Programming, which we consider appropriate for primary pupils, can be naturally divided into three domains (while first domain should proceed the other two, we believe that the second and the third ones can be implemented in any order or even in parallel). They are:

- Solving problems and handling solutions.
- Controlling an agent.
- Tinkering with interactive environments.

For each domain we have presented its main learning goals, corresponding computational concepts, computational practices, and essential cognitive operations to be performed; selection of activities and examples, which in detail illustrate various types of tasks and problems to be solved; several software applications that are being used;

and also several pedagogical observations and recommendations, which resulted from our collaboration with the primary teachers.

Most of the programming environments, which are being used at Slovak primary schools, are free applications, usually small microworlds focused on one of the domains listed above, and one or several cognitive operations belonging to that domain. As partly validated in chapter 4, teachers usually exploit environments which they find attractive (although often not being able to verbalize which criteria they apply for judging this). However, they clearly favour environments supplemented with teacher materials and activities for pupils, and environments, which they may give away to their pupils for their home work and play.

Our experience in implementing programming at the lower secondary stage ISCED 2 (although not based on systematic evidence yet) shows that three domains presented in the paper for primary stage can seamlessly be picked up and further elaborated in lower secondary years to cover further cognitive operations (like conditional steps in programs, abstractions, i.e. procedures without or with parameters etc.). However, extensive research to help us better understand cognitive demands of such programming and real values of educational programming for the complex development of primary and secondary students is inevitable. We have already undertaken some initial steps in this direction, see e.g. (Gujberova and Kalaš, 2013).

As we document in chapter 1, informatics in upper secondary education has considerably long tradition in Slovakia. In recent years, it has been extended as a mandatory subject to lower secondary level (2005) and primary level (2008). In chapter 2, we briefly characterized its curriculum and its learning goals and especially the key role of programming within the subject.

We fully focused on educational primary programming in the paper. In chapter 3, we presented in detail our approach to such programming together with corresponding computational concepts, cognitive operations, and programming environments employed in our classes. In chapter 4, we then described how the CPD for primary in-service teachers has been implemented – with partial successes and numerous obstacles and challenges that require permanent and intense support from the institutions responsible for education. In spite of many obstacles and slow progression, there are many positive and stimulating reactions from primary teachers who implement elementary informatics with exceptionally positive involvement. They also report positive attitudes of their pupils.

The development of the subject of informatics in primary school is a long-term process. In it, we must thoroughly respect the requirements of the developmental appropriateness, carefully observe and analyse the needs of the pupils, respect all stages of their learning processes, set correct priorities, and apply proper tools – so that we support the development of such programming, which our pupils will clearly benefit from. In this aspect, we deeply agree with Papert, Ackermann and other seminal authors when they advise not to learn programming for the sake of programming. Instead, we should...

> *use the knowledge of programming to create contexts where other playful learning can happen. Children will engage in programming if they can get something out of it right now – not later when they'll grow up,* (Ackermann, 2012).

References

Ackermann, K.E. (2012). Programming for the Natives: What is it? What's in it for the Kids? In: Kynigos, Ch., Clayson, J., Yiannoutsou, N. (Eds.), *Proceedings of Constructionism, Athens, Greece August 2012*. National & Kapodiststrian University of Athens, Athens, 1–10. Updated version obtained via CRN Japan: http://www.childresearch.net/papers/pdf/digital_2012_03_ACKERMANN.pdf

Barr, V., Stephenson, Ch. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads*, 2(1), 48–54. DOI: 10.1145/1929887.1929905 http://doi.acm.org/10.1145/1929887.1929905

Blackwell, A.F.(2002). What is Programming? In: Kuljis, J., Baldwin, L., Scoble, R. (Eds.), *Proceedings of 14th Workshop of the Psychology of Programming Interest Group*. Brunel University, 204–218.

Blaho, A., Kalaš, I., Tomcsanyiova, M. (1995). Experimental curriculum of informatics for 11 year old children. In: *WCCE'95 Liberating the Learner: Proceedings of the sixth IFIP World Conference on Computers in Education*. Chapman & Hall, London, 829–841.

Blaho, A., Salanci, L. (2011). Informatics in primary schools: visions, experiences, and long-term research prospects. In: Kalaš, I., Mittermeir, R. (Eds.), *Informatics in Schools: Contributing to 21st Century Education*. LNCS 7013, Springer, 129–142. ISBN 978-3-642-24721-7.

Brennan, K., Resnick, M. (2012). *Using artefact-based interviews to study the development of computational thinking in interactive media design*. Paper presented at annual American Educational Research Association meeting. Vancouver, BC, Canada.

Dagienė, V., Stupurienė G. (2016). Bebras – a sustainable community building model for the concept based learning of informatics and computational thinking. *Informatics in Education*, 15(1), 25–44.

Eisenberg, M., Elumeze, N., MacFerrin, M., Buechley, L. (2009). Children's programming, reconsidered: settings, stuff, and surfaces. In: *Proceedings of the 8th International Conference on Interaction Design and Children (IDC '09)*. ACM, New York, NY, USA, 1–8. DOI:10.1145/1551788.1551790 http://doi.acm.org/10.1145/1551788.1551790

Gibson, J.P. (2003). A noughts and crosses Java applet to teach programming to primary school children. In: *Proceedings of the 2nd International Conference on Principles and Practice of Programming in Java (PPPJ '03)*. Computer Science Press, Inc., New York, NY, USA, 85–88.

Gujberova, M., Kalaš, I. (2013). Designing productive gradations of tasks in primary programming education. In: Proceedings of the 8th Workshop in Primary and Secondary Computing Education WiPSCE '13. ACM, New York, NY, USA, 108–117. DOI: 10.1145/2532748.2532750 http://dl.acm.org/citation.cfm?id=2532750

Hu, Ch., (2011). Computational thinking: what it might mean and what we might do about it. In: *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education (ITiCSE '11)*. ACM, New York, NY, USA, 223–227. DOI:10.1145/1999747.1999811 http://doi.acm.org/10.1145/1999747.1999811.

Hubwieser, P. (2012). Computer science education in secondary schools – the introduction of a new compulsory subject. *ACM Transactions on Computing Education*, 12(4), Article 16 (41 pages). DOI:10.1145/2382564.2382568 http://doi.acm.org/10.1145/2382564.2382568

Informatics Europe and ACM Europe (2013). *Informatics Education: Europe Cannot Afford to Miss the Boat*. Report of the joint Informatics Europe & ACM Europe Working Group on Informatics Education. http://www.informatics-europe.org/images/documents/informatics-education-europe-report.pdf.

Kalaš, I. (2010). *Recognizing the Potential of ICT in Early Childhood Education: Analytical Survey*. UNESCO Institute for Information Technologies in Education, Moscow. ISBN 987-5-905175-03-9.

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J., Werner, L. (2011). Computational thinking for youth in practice. *ACM Inroads*, 2(1), 32–37. DOI:10.1145/1929887.1929902 http://doi.acm.org/10.1145/1929887.1929902.

Lu, J.J., Fletcher, G.H.L. (2009). Thinking about computational thinking. *ACM SIGCSE Bulletin*. DOI:10.1145/1539024.1508959.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., Eastmond, E. (2010). The scratch programming language and environment. *ACM Transactions on Computing Education*, 10(4), Article 16 (15 pages). DOI:10.1145/1868358.1868363 http://doi.acm.org/10.1145/1868358.1868363

Mayerova, K., Veselovska, M. (2016). How to teach with LEGO WeDo at primary school. In: *Proceedings of 7th International Conference on Robotics in Education (RiE 2016)*, Vienna. To be published in the Springer Series: Advances in Intelligent Systems and Computing.

Mogardo, L., Cruz, M., Kahn, K. (2006). Radia Perlman – a pioneer of young children computer programming. In: *Current Developments in Technology-Assisted Education*, 1903–1908. CiteSeerX: 10.1.1.99.8166. http://citeseerx.ist.psu.edu/viewdoc□/summary?doi=10.1.1.99.8166

National Curriculum, Informatics for ISCED 1 (2011, 2015). (In Slovak). http://www.statpedu.sk/sites/default/files/dokumenty/inovovany-statny-vzdelavaci-program/informatika_pv_2014.pdf

Papert, S. (1999). What is Logo and who needs it. In: *Logo Philosophy and Implementation.* Highgate Springs, Vermont: Logo Computer Systems Inc. ISBN 2-89371-494-3.

Pekarova, J. (2008). Using a programmable toy at preschool age: Why and How? In: *Workshop Proceedings of SIMPAR 2008, International Conference*. 112–121. ISBN 978-88-95872-01-8.

Piaget, J., Inhelder, B. (1993). *La Psychologie de l'enfant*. Paris: Presses Universitares de France.

Polya, G. (2004). *How to Solve It.* Princeton University Press.

Przybylla, M., Romeike, R. (2014). Physical computing and its scope – towards a constructionist computer science curriculum with physical computing. *Informatics in Education*, 13(2), 241–254.

Resnick, M. (2012). Point of view: reviving papert's dream. *Educational Technology*, 52(4), 42–64.

Selby, C.C. (2012). Promoting computational thinking with programming. In: *Proceedings of the 7th Workshop in Primary and Secondary Computing Education (WiPSCE '12)*. ACM, New York, NY, USA, 74–77.

Selby, C.C. (2013). Computational thinking: the developing definition. In: *ITiCSE Conference 2013*, University of Kent, Canterbury, England (e-prints), 6 p.

Settle, M., Franke, B., Hansen, R., Spaltro, F., Jurisson, C., Rennert-May, C., Wildeman, B. (2012). Infusing computational thinking into the middle- and high-school curriculum. In: *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE '12)*. ACM, New York, NY, USA, 22–27.

Tomcsanyiova, M., Tomcsanyi, P. (1997). Experimental IT education for lower secondary school using Windows and Comenius LOGO. In: *Learning and Exploring with LOGO: Proceedings 6th European LOGO Conference*. Budapest, 263–272. ISBN 963-8431-91-1. http://eurologo.web.elte.hu/lectures/tomcsa.htm

Tomcsanyi, P. (2012). Small interactive computer activities made by primary teachers. In: *Information and Communication Technology in Education 2012*. Ostrava: University of Ostrava, 263–272.

Vanicek, J. (2013). Introducing Topics from Informatics into Primary School Curricula: how do teachers take it? In: Diethelm, I., Arndt, J., Dunnebier, M., Syrbe, J. (Eds.), *Informatics in Schools: Local Proceedings of the 6th International Conference ISSEP 2013, Oldenburg, Germany – Selected Papers*. Universitätsverlag Potsdam, 41–51.

Wing, J.M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A*, 366, 3717–3725. DOI:10.1098/rsta.2008.0118

Wing, J.M. (2011). Research notebook: computational thinking – what and why? In: *The Link*. Pittsburgh, PA: Carnegie Mellon University.

Links to Microworlds and Programming Environments

Baltie: http://www.sgpsys.com/en/whatisbaltie.asp
Bee Tasks: see (Gujberova and Kalaš 2013), for the microworld itself contact the authors
Cards Tool: http://edi.fmph.uniba.sk/~tomcsanyi/Karticky/
EasyLogo: http://www.salanci.sk/EasyLogo/index.html
LEGO WeDo: http://www.legoeducation.us/eng/product/lego_education_wedo_software_v1_2_and_activity_pack/2239
Microworlds JR: http://www.microworlds.com/solutions/mwjunior.html
Scratch: http://scratch.mit.edu/
Thomas the Clown: http://www.r-e-m.co.uk/logo/?Titleno=7485

M. Kabátová, was an assistant professor of informatics education at Faculty of Mathematics, Physics and Informatics at Comenius University, Bratislava. Her research interests included elementary programming, educational robotics and qualitative research methodology applied to educational research in informatics education. She is a co-author of several research papers and learning materials dealing mostly with educational robotics. Since 2014 she runs an SME distributing cochlear implants in Slovakia and provides services for cochlear implants users. Currently she is planning to initiate a workshop providing a learning space for children with or without cochlear implants where they could explore programming and building autonomous LEGO robots.

I. Kalaš is a professor of informatics education at Comenius University, Bratislava. His professional interests include development of constructionist educational interfaces for learning for children and research in the field of the impact of digital technologies on learning. Ivan is a co-author of several programming environments for children, including SuperLogo, Imagine Logo, Thomas the Clown and RNA (Revelation Natural Arts) adopted by thousands of schools, home and abroad. He is also an author or co-author of several books and textbooks on children programming and informatics, which have been published in several languages and countries in Europe and beyond. He has also been active in several national and international policy efforts and initiatives. Ivan represents Slovakia in the IFIP Technical Committee for Education. From 2008 to 2013, he was a member of the International Advisory Board of the 'Microsoft Partners in Learning' initiative. From 2014 he is a visiting professor at UCL Knowledge Lab, London.

M. Tomcsányiová is an assistant professor of informatics education at Comenius University, Bratislava. She is a guarantor of a bachelor study programme for future teachers of informatics. She reads the courses on programming and educational aspects of programming and conducts corresponding research in the field of educational programming in all levels of education. Monika is a co-author of several textbooks and methodical teacher materials. She is also involved in designing small educational software environments supporting informatics education in Slovak primary and lower secondary schools. She develops tasks and organizes informatics challenges for primary and secondary students, including Imagine Logo Cup, Scratch Cup and national Bebras. Her area of research is didactics of programming for lower secondary schools. She is a co-author of research papers in this area.