Environments for programming in primary education

Monika Gujberová and Peter Tomcsányi

Department of Informatics Education, Faculty of Mathematics, Physics and Informatics, Comenius University, 842 48 Bratislava, Slovakia

{gujberova, tomcsanyi}@fmph.uniba.sk

Abstract. The aim of our article is to collect and present information about contemporary programming environments that are suitable for primary education. We studied the ways they implement (or do not implement) some programming concepts, the ways programs are represented and built in order to support young and novice programmers, as well as their suitability to allow different forms of sharing the results of pupils' work. We present not only a short description of each considered environment and the taxonomy in the form of a table, but also our understanding and opinions on how and why the environments implement the same concepts and ideas in different ways and which concepts and ideas seem to be important to the creators of such environments.

Keywords: Primary informatics, Programming environments for children, Comparing programming environments

1 Introduction

At the beginning of our work there was an idea to look at the programming environments used in Slovak schools from a technical point of view, from the point of view of a developer of such environments rather than from the point of view of their users. Our interest was based on our previous experience with developing such environments. We wanted to know the ways they implement (or do not implement) some programming concepts, the ways programs are represented and built in order to support young and novice programmers, as well as their ability to enable different forms of sharing the results of pupils' work.

The second section shortly describes the considered environments that were chosen based mainly on our knowledge of their widespread use in Slovak schools.

The third section shortly describes how we chose our initial set of criteria and how we iteratively enhanced them. Then it presents our final list of features as well as the knowledge that we gained in the process of studying the environments. Finally we present a grid of features for the selected ten environments in a form of a table.

2 The Environments

For our study, we chose at first eight environments, knowing from our contact with schools and our in-service courses for primary teachers that they are actually wide-

spread in Slovak schools (sections 2.1 and 2.2). Later we added two environments that are not yet commonly used in Slovakia, but we consider them worth comparing to the other environments as well as to present them to our teachers (section 2.3).

Our choice includes not only worldwide well-known environments (like Scratch or Alice), but also less known ones (like Karel or Baltie) and others specific to our country (like The Jumper or Phillip the Ant).

2.1 Logo descendants

Logo has been developed not only for children, but also for teachers [1]. First implementations were text-based. The turtle came later, at first as a physical robot turtle, later as the on-screen light turtle [2].

Imagine Logo is a full implementation of Logo developed in Slovakia and used in several countries [3]. It supports pure textual programming, but parts of the programs (especially parameters of standard procedures) can be constructed by selecting from lists, and its visual objects can be constructed by direct manipulation.

Scratch is a well-known modern descendant of Logo created by the Lifelong Kindergarten Group at the MIT Media Lab [4]. It has been localised to Slovak [5]. It uses puzzle-like construction of programs. The program is constructed from jigsaw puzzle-like pieces containing commands and parameters. The pieces either fit together or do not fit together depending on their shape.

EasyLogo (**Izy Logo** in Slovak language) [6] is a simplified turtle graphics program that has been implemented in a form accessible to pupils in primary education. It limits the turns to multiples of 45 degrees and redefines the metrics of the screen so that fd 1 always moves to the next grid point. The programs are constructed by dragging the icons of actions into the program. The parameters are set by choosing them from lists. It allows defining one's own new commands without parameters.

Lively picture is another simplified version of Logo. It has been implemented in Imagine Logo. It focuses on a widespread simple Logo activity – constructing *lively pictures*, which are stories or animations. They contain static or moving graphical objects. Some of them may react to user interaction (clicking or dragging by mouse). Its programming language is purely iconic, only the parameters of some commands are expressed in numbers.

2.2 Programming specialised robots

Karel 3D for Win32 is the most widespread implementation in Slovakia**.** Karel was originally developed as an introductory programming language for university students [7]. The robot can move and sense walls. It can put and collect marks (beepers), in newer implementations it can also put and collect bricks. The programs are written in text form, it includes procedures. It has no variables. Karel became popular in Czechoslovakia in the late 80s and it has been re-implemented for several kinds of computers.

Baltie (**Baltík** in Czech and Slovak) is an iconic programming language that originated in Czech Republic and is quite popular in the Czech Republic as well as in Slo-

vakia. The main character is a young magician Baltie, who can move around a 2D scene and conjure objects (square pictures). There are three modes of operation: creating the scene, direct mode conjuring and programming.

The Jumper [8] can jump up and down, step left and right and can repeat a list of statements. The pupils construct programs that lead the Jumper from its starting position to the doors leading to the next level by clicking on icons representing commands. The program must first be fully constructed and only then can it be run to see whether it is correct. The number of statements is limited so that the pupil must recognise repeated actions.

Phillip the Ant has to find a path through a maze, collect candies, avoid spider nets, collect other useful things and move things to their correct places. Phillip can be controlled in three modes: direct mode, arrows mode and iconic mode. The environment has been created in Imagine Logo as part of a master thesis [9], later extended for the use in in-service training courses in Slovakia [8].

2.3 Advanced environments

Microsoft Kodu Game Lab (Kodu) [10] has been designed for creating and playing one's own games. The whole program is event-based – it is a sequence of *When something happens do something* statements. The conditions and commands are expressed by icons selected from menus (repositories). The Web page of Kodu contains many tutorials that help to learn its handling.

Alice 3D is a programming environment that allows creating animations, interactive stories, videos and games. Originally it has been developed for teaching the basics of programming especially targeting pupils with no pre-knowledge in programming or little interest in maths. Another goal was to attract more girls to programming [11].

3 The features

We started to observe the ten environments using the criteria derived from the extensive work of Kelleher and Pausch [12], even if the goal of their work has been somewhat different from ours. Then we iterated these three steps: (1) studying the environments and coding their properties according to the existing set of criteria, (2) gaining knowledge about the properties of the environments, and (3) reflecting the gained knowledge by modifying the set of criteria to better meet our specific goals. This section presents the final set of criteria. We describe them in detail and give examples of the environments that meet or do not meet the specific criteria. Table 1 presents the complete grid of criteria and environments.

3.1 General vs. Specialised

Some of the considered environments are strictly *specialised* microworlds, like The Jumper and Phillip the Ant. Other environments include *general* programming lan-

guages like Imagine Logo, Scratch, Kodu and Alice. In primary education both these types of environment are useful for the teacher even if they serve different purposes.

Specialised environments usually include some number of prepared tasks and it is possible for the teacher (or even for pupils) to prepare their own tasks within the bounds of the environment. Sometimes it is possible to do this inside the environment in an interactive and visual way. We coded such environments as *internal* in the row *creating tasks*. Sometimes an external program is needed (usually a plain text editor), coded as *external*.

3.2 Programming style

For our purposes the programming style is the way of thinking while programming. The considered environments use three programming styles: *procedural*, *object-oriented* and *event-based*. The procedural style uses lists of statements that are executed from the beginning to the end. Most of them also allow the users to define their own procedures and/or functions. The *object-oriented* style uses the abstraction of object which is very common in professional programming environments. The environments for children often include partially implemented object-orientation, which we code as the *level of object orientation* in a next criterion.

The *event-based* style allows defining snippets of code that run when an event happens. Typical events are mouse clicks and drags, keyboard key pushes/releases or collisions of objects.

3.3 Level of object orientation

Blaho and Kalaš [3] defined a hierarchy of several levels of using object-oriented features. For our purposes we slightly modified their hierarchy as follows:

- *objects* – existence of objects as entities that encapsulate data (internal state) and procedures (methods) that can change the data
- *cloning* – a command or a user interface action that allows the user to create an identical copy of a complete object
- *own variables* – the users can define their own state variables in objects
- *own methods* – the users can define their own methods in objects that may override the standard behaviour of the object
- *inheritance* – one object can serve as a parent of another object that inherits all the behaviour of its parent (and may override some behaviour by defining its own methods)
- *classes* – special entities that serve only as parents (or prototypes) for creating objects (instances of the class) and themselves are not valid objects
- *multiple inheritance* – one object can inherit behaviour from several parent objects or classes.

One more criterion is outside the hierarchy – whether the programming language allows to *programmatically create and destroy* objects or instances of classes. Kodu and Imagine allow it while Alice and Scratch do not.

3.4 Programming Constructs

- *conditional* (*if* statements)
- *count loop* – a loop with a known number of repetitions but without an explicit loop variable (like *repeat* in Logo without using the *repcount* function)
- *for* loop
- *while* loop
- *variables* (global, local or object variables)
- *own procedures or methods*
- *parameters* in own procedures or methods

3.5 Code Representation

We consider the representation being *textual* only if it can be fully edited letter by letter. The other two representations are *card or puzzle-like* and *iconic*. The former still uses text to represent the program, but parts of the text are written on either rectangular cards or puzzle-like pieces, while the latter uses a purely graphical way of expressing the commands (even if sometimes it still needs numbers).

Some environments use two representations. EasyLogo represents the basic actions (commands) as icons that the user must drag into the program, but after one such icon is dragged, it changes into a textual card that shows the name of the command.

3.6 Project Construction

Modern programming environments for children tend to minimise typing or try to abandon it completely. However, an interesting research study suggests that it is not easy to predict whether these kinds of simplification will really make the learning of programming easier or not [13]. We distinguish six types of project construction:

- *typing* code
- *selecting* pieces of code from menus or similar user interface elements
- assembling code from *rectangular cards*
- assembling from *puzzle-like pieces*
- assembling from *icons*
- *direct manipulation* for creating (visual) objects

Imagine Logo allows typing, but specific parts of the code can also be selected from menus and special helper dialogs (choosers). Additionally objects (e. g. turtles) can be created by direct manipulation, too. Karel allows constructing the program only by typing, but the environment of the robot can be created by direct manipulation. Scratch and Alice construct code by assembling from pieces.

3.7 Preventing (syntax) Errors

Alternative ways of program representation and/or construction can usually prevent syntax errors. Sometimes also semantic errors can be avoided, e.g. Scratch does not allow the user to place a number in a place where a Boolean expression belongs.

We distinguish three approaches: *shape matching* (the shapes of pieces must match, otherwise a part of the program cannot be constructed), *selection from valid options* (a list of valid options is presented as a menu, pop-up window, a pile of selectable cards etc.), and *syntax directed editing*.

Syntax directed editing recognises the structure of the program and allows the user to drag/select only complete structures (e. g. complete *if-then-else* command or *while* loop), thus not allowing the user to make errors by forgetting a part of the command.

3.8 Saving and Exporting

Being able to show the results of their work to others is an important point for children. Some environments are able to export the whole project or at least its result in a form that can be run or at least shown, using only standard software with no need to install the environment itself. Sharing the result of one's work on the Web is another important feature. We distinguish the following features of the environments:

- *own format* – whether it is possible to save the project into a file that can be later loaded by the same environment and used further
- *result* – whether the environment can save only the resulting static picture or series of pictures (video or animation) that can be shown later using a player of a standard file format without the ability to continue the work on the project
- *standalone project* – whether it is possible to create a standalone executable file that can be run without the need of having the whole environment available
- *Web site* – whether the creators of the program provide a Web site to share the programs created by pupils
- *save for Web* – whether it is possible to save the resulting program in a form that can be run inside a browser, in which case we list the additional software that is needed (except the browser itself) to run it.

3.9 Localization

In primary schools it is very important to present the computer programs to pupils using their native language. Several environments on our list originate from Slovakia, other environments can be localised *by the end-users* (Scratch and Kodu), and for some others several localisations already exist. In our table we also present the *number of localizations* that we know about.

Table 1. The resulting grid of features and environments

		Imagine Logo	EasyLogo	Lively picture	Scratch	Karel 3D for Win32	Baltie 3	The Jumper	Filip the Ant	Kodu	Alice 2.2
General vs. Specialised	specialised		x	x		x	x	x	x		
	way of defining the tasks		ext			int	int	ext	int		
Programming style	procedural	x	x			x	x	x	x		
	object-oriented	x								x	x
	event-based	x		x	x					x	x
Level of Object Orientation	objects	x	x	x	x					x	x
	cloning	x		x	x					x	
	own variables	x			x						
	own methods	x								x**	x
	inheritance	x									x
	classes	x								x	
	multiple inheritance	x									
	programatically create/destroy	x								x	
Programming Constructs	conditional	x			x	x	x			x	x
	count loop	x	x	x	x	x	x	x			x
	for loop	x			x		x				x
	while loop	x			x	x	x				x
	variables	x			x		x			x	x
	procedures/methods	x	x		x***	x	x				x
	parameters	x	x								x
Code Representation	textual	x				x					
	card or puzzle-like		x		x			x			x
	iconic		x	x			x	x	x	x	
Project Construction	typing	x			x****	x					
	selecting	x	x	x	x			x		x	x
	rectangular cards										x
	puzzle-like pieces				x						
	icons		x	x			x	x	x	x	
	direct manipulation	x		x	x	x				x	
Preventing (syntax) Errors	shape matching				x						
	selection from valid options		x	x	x		x	x	x	x	x
	syntax directed editing		x	x	x					x	x
Saving and Exporting	own format	x	x	x	x	x	x		x	x	x
	result	anim. gif gif, jpg	wmf		gif						mov jpeg
	standalone project	exe									
	Web site				x					x	
	save for Web	plugin									Java 3D
Localisation	by the end user				x					x	
	number of localisations	7	4	1	50	4	2	1	1	7	2*

Legend:

* Applies to Alice 2.3

** Kodu has no real methods, but object can define their own reactions to events

*** Only with an extension called BYOB (Bring your own blocks)

**** Typing is very limited in Scratch – e.g. number ant text data

4 Conclusion

Our work started as a trial to make a systematic list of environments useful for primary education in our country. After a few iterations we found the result interesting not only for us, but also for a broader audience. The list itself shows the variability of the environments from specialised ones to general ones, from those inspired by Logo to others allowing the user to create complex videos, from textual programming through puzzle-like programming to iconic programming.

We believe that the information is valuable both to developers of such environments and their users. The developers get a new overview of all possible approaches used, while the teachers get acquainted with environments they did not yet know about or get more information about environments that they have already heard about.

References

1. Feuerzeig, W. et al.: Programming-Languages as a Conceptual Framework for Teaching Mathematics. Final Report on the First Fifteen Months of the LOGO Project: Bolt, Beranek and Newman, Inc., Cambridge, MA (1969)
2. Papert, S. et al.: Logo Philosophy and Implementation, LCSI (1999)
3. Blaho, A., Kalaš, I.: Object Metaphor Helps Create Simple Logo Projects. In Proceedings of EuroLogo 2001. Edited by G. Futschek. Linz, August. pp. 39–43 (2001)
4. Maloney, J., Resnick, M., Rusk, N., Silverman, B., Eastmond, E.: The Scratch Programming Language and Environment. In ACM Transactions on Computing Education, Vol. 10, No. 4, Article 16 (2010)
5. Drahošová, M.: Úvod do programovania v prostredí Scratch. Master thesis FMFI UK, Bratislava, Slovakia, (2010) [Introduction to programming in Scratch. Master thesis, Comenius University, Bratislava. In Slovak language]
6. Salanci, L.: EasyLogo – discovering basic programming concepts in a constructive manner. In: Proceedings Constructionism 2010, Paris (2010)
7. Pattis, R. E.: Karel – the robot, a gentle introduction to the art of programming. Wiley, London (1981)
8. Tomcsányiová, M. a kol.: Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika – Riešenie problémov a základy programovania 1. 1. Vydanie (2009) [Problem solving and programming basics, Textbook for an in-service teacher training course. In Slovak language]
9. Ondková, J.: Detský programovací jazyk Mravec pre 1. stupeň ZŠ. Diplomová práca. Bratislava: FMFI UK (2006) [Children's programming language The Ant for the primary school. Master thesis, Comenius University, Bratislava. In Slovak language]
10. http://kodu.blob.core.windows.net/kodu/Curriculum_MARS/Level%202%20-%20Search%20and%20Explore%20Mars.pdf (last checked 1/31/2013)
11. Utting, I. et al.: Alice, greenfoot and scratch – A discussion. [online] ACM Transactions on Computing Education, Vol. 10, No. 4, Article 17 (2010) http://www.cs.kent.ac.uk/pubs/2010/3071/content.pdf (last checked 1/31/2013)
12. Kelleher, C., Pausch, R.: Lowering the barriers to programming: a taxonomy of programming environments and languages for novice programmers. ACM Computing Surveys, 37(2), pp. 88–137, (2005)
13. Lewis, C. M., "How Programming Environment Shapes Perception, Learning and Goals: Logo vs. Scratch", Proc. SIGCSE'10, ACM Press, WI, USA, (2010)