The role of programming in the pre-service teacher training

Andrea Hrusecka

Comenius University, Faculty of Mathematics, Physics and Informatics

Mlynska dolina, Bratislava

hrusecka@fmph.uniba.sk

Abstract

In the pre-service teacher training (for lower and upper secondary education) at the Faculty of Mathematics, Physics and Informatics at Comenius University, Bratislava we have implemented a six-term course on Information and communication technologies and computer literacy. Each term of the course is focused on a particular aspect of ICT (like Basics of ICT, Computer systems, Algorithms development and programming, Educational software in subjects etc.).

In our paper we present the contents and strategy of the third and the fourth terms of the course, namely The Basics of algorithmic skills and programming for future teachers. The goal of these seminars is to develop algorithmic skills and to learn how to build simple microworlds in Imagine Logo. We briefly list all topics and concepts of the seminars, all key terms from the microworlds development and the programming techniques we are working with. Finally, we present one more complex Imagine project, which our students are asked to develop by themselves at the end of the course.

Keywords

Pre-service teacher training, information literacy, ICT, algorithmic skills, Imagine Logo

1. Introduction

At the Department of Informatics Education we developed a framework of a new course of 5 terms titled *Information and Communication Technologies for Future Teachers.* From the academic year 2001/2002, it has been implemented into the curriculum for all teacher students as a compulsory course. Within this course we would like to improve their information literacy. As far as we find it necessary to apply different approaches in building ICT literacy of Informatics teacher students and other future teachers, in fact we have implemented two separate courses for them. Two lessons are allocated for this course per week.

In our contribution we will introduce the third and fourth term for **non-informatics teacher students**. The name of this part of our course is *The basics of algorithmic skills for future teachers* and the platform being used is Imagine Logo.

2. Conception of courses

At the department we are convinced that an important component of the ICT literacy is elementary algorithmic skills. This assumption is confirmed by several experiments, for example with the *Visual Fractions*, see (Lehotska & Kalas 2005). Experiments were run with teachers and with students as well. The result shows that *the development of the fractions activities (by teachers or students) is always limited by the level of user's algorithmic skills*

(Kalas 2005). When we talk about the development, we, first of all, mean the richness and complexity of the activities.

We try to develop the algorithmic skills of our students mainly in a very playful way (by building simple visual activities, creating lively pictures, easy games, etc). Our goal is to strengthen their abstract thinking, to develop their ability for discovering, analyzing and creating methods for problem solving, and to improve their creativity.

In the first run (in 2001/2002), the course was designed in the environment of Comenius Logo. From the next year on, we have changed the environment to more attractive Imagine Logo, which is designed for creating visual, open and interactive activities also for real beginners.

The lessons are being held in computer rooms. At the beginning of every lesson the students are given a short assignment from the material of the previous lesson. Usually they have to define a command or they have to answer a series of questions or they solve a problem on computer. After that, they receive a worksheet with definitions of new terms, with the interpretation of new activities and a list of complementary tasks. It means that the students do not have to take any notes during the lessons, so they can concentrate solely on the tasks and problems to be solved. At the end of the term every student has to solve a complex problem on computer (see chapter 4).

3. Study materials for the courses

By collecting the materials on every topic students get a complete study literature. These materials were published in the UK in a modified version by Logotron, Cambridge, titled *The Great Big Imagine Logo Project book* (Kalas & Hrusecka, 2004).

In the following part we characterize the topics of the lessons.

short contents	new terms	Imagine-Logo constructions
LogoMotion – Image Editor For Imagine 1		
► basic drawing, one picture ► working with a region (selection) ► working with photography	setting the colour (**HSV**, **RGB**, Palette), anti-aliasing, colour transparency, selection	
Basic Turtle Commands		
► basic movements, cleaning the page ► pen of the turtle, pen colour, pen width ► choosers, buttons window, random values ► more commands ► **repeat** command ► **edit** command	command line, command, instruction, input, variable, chooser, random input (**any**)	**forward** *num*; **back** *num*; **right** *angle*; **left** *angle*; **clean**; **cs**; **penUp**; **penDown**; **setPenColour** *colour*; **setPenWidth** *penWidth*; **setXCor** *X-cor*; **setYCor** *Y-cor*; **setPos** *point*; **setHeading** *angle*; **point** *diameter*; **repeat** N [*to_do to_do …*]; **edit** *nameOfCommand*; **repeat** 5 [**fd** 100 **rt** 720/5]

Commands With Variables 1

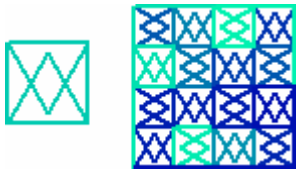
- defining your own commands
- commands with variable

Explore window, value of variable, random number, local variable

```
repeat N
  [ownComand to_do];
random number;
let "name value
```

Commands With Variables 2

- commands with several variables
- **poly** command
- pen colours and pen widths as variables



Events. Drawing While Dragging

- automatic dragging
- your own button – **onPush** event
- turtle and its **onDrag** event
- your own command as a reaction to **onDrag** event
- how to make command more random

Change t1 dialogue box, settings, auto dragging, button, event, range for turtle (**Window**)

```
pick [yellow6 yellow7
      yellow8 yellow9
      yellow10];
wait 30
```

Conditions

- generating random numbers: examples and exercises
- drag the flower and stamp
- reaction depending on the colour of the background – conditional instructions
- infinite walk – infinite tail recursion

scale, paint bar, conditional instruction, recursion, tail recursion, **Stop all**

```
sh (10 * random 2) - 5;
hideMe; showMe; stamp;
setShapeScale number;
dotColour;
if <condition>
   [what_to_do]
```

Random Walks. How To Terminate Recursion

- movements restricted into a rectangle
- compound conditions – **and**, **or**
- random walk: exercises

Stop Command Line

```
and logicalValue
    logicalValue,
(and logicalValue … ),
or logicalValue
   logicalValue,
(or logicalValue … ),
if <condition> [stop]
```

Processes

- parallel process as an "engine"
- processes with names, cancel the process
- page and pane
- horizontal slider
- build your own game – connecting dots

"engine" – process, switch button (events **onPush**, **onRelease**), named process, cancel process, page, pane, slider

```
every milSec [to_do];
p1'clean;
(every milSec [to_do]
        "nameOfProcess);
cancel "nameOfProcess;
after milSec [to_do]
```

Multiple Turtles 1

- **New Turtle** tool
- how to address turtles
- addressing one turtle with apostrophe
- in-the-middle curve, position as a vector, computations with vectors

cloning turtles, copy to, paste from clipboard, apostrophe notation

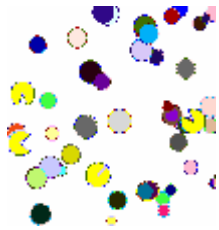
```
ask "t1 [to_do … ];
ask [turtleA turtleB … ]
   [to_do … ];
all;
askEach [turtleA turtleB…]
   [to_do … ];
```

Image As A Turtle Shape

- exploring default shape, **stamp** command
- image, frames and frame items. Pacman
- In the Meadow project

image as a value of a variable, frames and frame items, animation loop

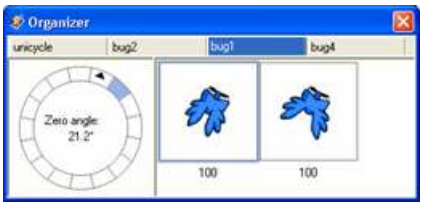
```
make "myShape shape; fill;
```



LogoMotion – Image Editor For Imagine Logo 2

- working with animation
- transform and generate
- create your own animation

heading and frame mode, hot spot, delay, animation loop, image transformations, onion skin



Multiple Turtles 2

- creating turtles with the **new** command, list of settings
- flocks of turtles
- turtles chasing

list of settings, permanent and temporary activity

```
new "Turtle [];
new "Turtle
  [setting value … ];
eraseObject all;
loadImage "nameOfFile;
butFirst all;
setHeading towards "t1;
```

turtles – pig hunting
- ▶ generating turtles while moving around a circle

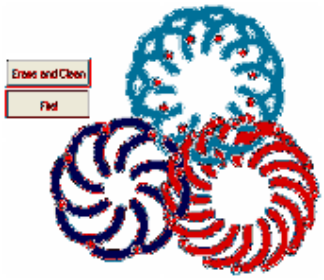
```
repcount;
ask [t1 t5 t7] [show who];
mousePos;
tell "nameOfTurtle
```

Problems And Projects

- ▶ firework
- ▶ gymnastic festival – many turtles in a regular formation
- ▶ bug contest
- ▶ more pig hunting strategies
- ▶ living picture – several objects and processes

creating and controlling many objects in parallel

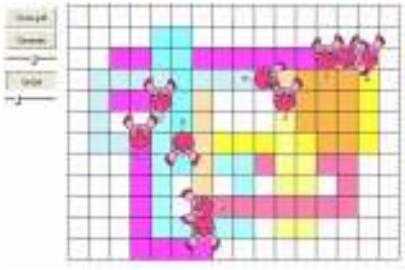
```
setFrame number;
reorder [turtle1 turtle2…]
```



Project 1 Bugs In A Sheep Pen

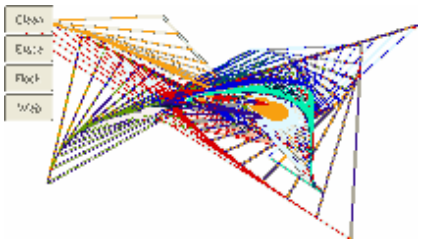
- ▶ midcourse project (draw a grid, generate N objects, start a process to make the bugs move, ...)

setEvent, **dotColourAt**



Revision. Problems And Assignments

- ▶ Meadow
- ▶ Heart – recognizing compound area
- ▶ Heart 2 – random walk restricted into a compound area
- ▶ Spider web, alternative pig hunting



Drawing Lists 1

- ▶ programmable shapes, drawing lists
- ▶ Mill project
- ▶ setting pen colour and pen width inside the drawing list

drawing lists, hot spot

```
setShape [commands];
ask lastName [commands]
```

Drawing Lists 2

- ▶ what can be used inside a drawing list

list of instructions, **onLoad** event

```
setShape [ ];
circle diameter;
filledCircle diameter;
```

- Arrow project
- Watching Eyes project, **onLoad** event
- The Watch project

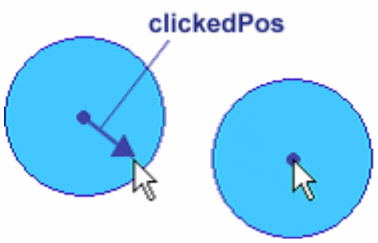
```
ellipse [num1 num2];
ellipse [num1 num2 a1 a2];
filledEllipse [num1 num2];
polygon drawing_list;
label text;
spline drawing_list;
outline list_of_points;
show time
```

Drawing Lists 3 Dynamic Programmable Shapes

- dynamic programmable shapes
- Circle project 1 – slider to resize the circle
- Circle project 2 – resizing by dragging the border
- introduction to dynamic geometry

object variable - setting, **onChange** slider's event, object oriented programming

```
clickedPos;
ifElse <condition>
  [commands] [commands];
setEvent <name> [commands]
```



Lists 1 Collecting Points And Re-drawing Them

- lists and their items
- empty list
- collecting points by mouse – simple solution
- processing lists of points
- collecting points as a property of the page
- collecting drawings with several segments

list,
empty list,
list of points,
list of lists of points

```
sh item 3 time;
sh count [20 30];
sh fPut 10 :L;
make "L fPut 10 :L;
make "Points
      lPut pos :Points;
eraseEvent <name>
```

Lists 2 Working With Texts. Sentences

- turtle printing into the page, command **label**
- printing into circles, polygons and columns
- **textSize** command
- text as a shape of the turtle
- generating random sentences

bitmap and vector font,
sentences as a lists of words,
text as a shape of the turtle

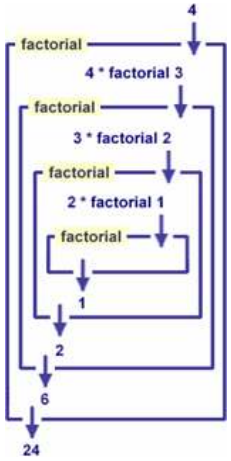
```
t1'setFont <F9>; textSize;
```

Lists 3 Operations. Building Lists From Parts

- commands and operations
- defining your own operations, **output** command
- recursive operations on numbers
- lists of turtle names

operations, recursive operations on numbers, trivial case

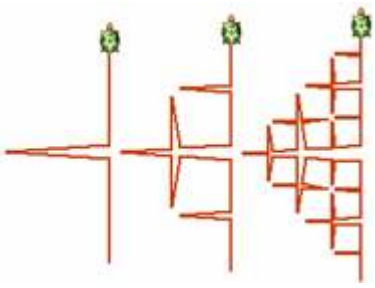
```
output;
ifElse mod :N 2 = 0
  [op "true][op "false];
op mod :N 2 = 0
```



Recursion To Draw Fractal Curves

- fractal curves, snow flakes and variations
- carpet fractal curves
- recursive trees
- recursive trees with random irregularities

level of recursion

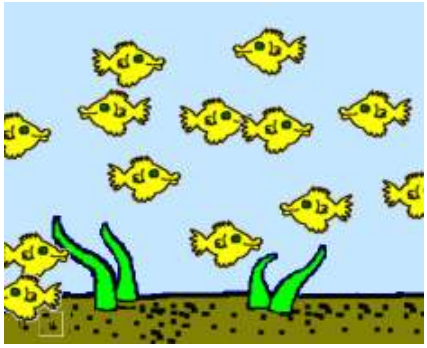


Classes And Object Oriented Approach 1

- load background
- **onClean** event of the page
- creating your own class (family), **newClass** command and **allOf** operation
- fish with their own speeds and sizes
- **Save as Web Project**...
- class of wooden blocks, its subclasses

classes of objects, load background, flat button, behaviour of the family (class), inheritance

```
setBGPicture "fileName;
newClass "Turtle "Fish
  [pen pu heading 90
   shape :fish];
allOf "Fish;
```



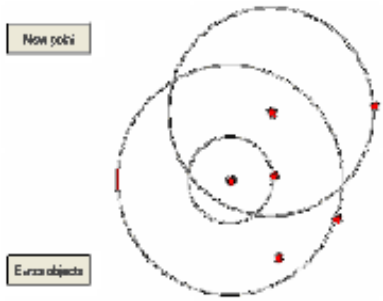
Classes And Object Oriented Approach 2

```
eraseObject myName
```

- grey stars on the sky
- stars moving away from the centre of the page, `onCreate` event
- lighting the stars
- improved dynamic geometry

`onCreate` event, object erases itself, common non-standard setting `namePrefix`

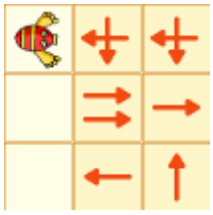
`eraseObject myName`



Project 2 Arrows Labyrinth

- final project (define Card and other classes, generate arrows labyrinth, define behaviour for the bug)

operation `is?`, overlapping turtles



4. Final project – The Arrows labyrinth

We will present the level and the style of the work we expect form our students after completing both 3rd and 4th terms of the course by demonstrating their final project.

At first, we show our solution of the project to the students and then we give them the detailed written assignment with instructions how to proceed. They build the project with *progressively constructing the scene and composing the objects' behaviours*.

The goal of the Arrows labyrinth project is to find the right way across the labyrinth – form the left side to the right. The `Bug`, which walks along labyrinth, has to respect rules, which derive from the arrow cards. The player wins, if he finds the way form the left to the right. He loses, if the `Bug` leaves the labyrinth up or down. The task for the students is to develop a project, which makes it possible to build and set the labyrinth and to play the labyrinth which means to try to find the right way through it.

The page size in activity is `800x630` and its origin is in point `[400 315]`. The first task is to generate the labyrinth. There are two buttons on the page. The first of them enables to generate the labyrinth, set the directions of the cards and test the way out from the labyrinth (`onPush` – `newLabyrinth` (own command), `onRelease` – `eraseObject all clean`). The second button freezes the modifications of the cards and allows only trying to solve the labyrinth (`onPush` – `startGame`).

The next step is to create the structure of the cards.

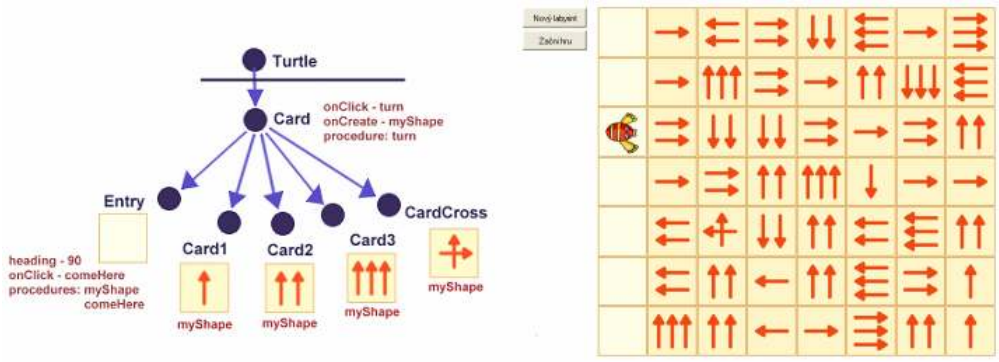


Figure 1. The structure of the card objects and the completed game itself

All versions of **myShape** command for the card classes have the same form. Size of each card is **69 x 69** and the hot spot is always in its centre.

The semi-empty structure of **myShape** definition is the part of the assignment.

```
to myShape
 setShape
  [pu bk 34 lt 90 pd
   setPc orange5 setFc <colour>
   polygon
     [repeat 4
       [fd 34 rt 90 fd 34]]
   ...]
end
```

The **turn** command (in the definition of **Card** class) makes the card turn smoothly by **90** degrees to the right in **45** steps with very small delay.

After click on the entry card, the **Bug** (being anywhere in the labyrinth) will turn towards this card, come here and turn into the labyrinth. The **comeHere** command is the part of the assignment too.

```
to comeHere
 let "myPos pos
 ask "P
  [setHeading towards :myPos
   while [abs pos - :myPos > 1]
         [fd 1 wait 5]
   setPos pos
   setHeading 90]
end
```

The **newLabyrinth** command first erases all objects and cleans the page. At the position of **[-196 210]** creates new turtle **Bug** with the following settings: **heading 90**, **rangeStyle bounce**, **font** arbitrary, **penColour** arbitrary, **pen pu**, shape from the file, **shown false**, **onClick** as global **go** command. Then the cards are generated with the help of this hidden **Bug**. The **Bug** by **repeat 7 […]** command, walks along seven lines of the labyrinth, it moves from centre of the card to the centre of the next card in **70** steps. In each line it first creates an entry card, then seven arrow cards. In the **new** command it chooses at random by the **pick** operation one of the families **[Card1 Card2 Card3 CardCross]** and creates a new card of that family at its actual position. For each new arrow card has to be set the heading at random from the list of options **[0 90 180 270]**. When the **Bug** generates all seven lines, it should move home, move **toFront** above all cards and show itself by **showMe**. The very last command of the **newLabyrinth** should be **Card'setEvent "onClick [turn]**. The **Card** class already has that event defined, but while solving the labyrinth, the cards can not turn, wherefore this event has to be erased. Then the **step** command has to be defined. This command makes the **Bug** move ahead from the centre of one card to the centre of the neighbouring card. At this stage of developing the project, the **go** command could be defined as **step**.

In order to complete the task, we have to modify the **go** command, also define the **startGame** command and auxiliary operation **family?**. This operation has two inputs: the name of a card and the name of a family-class. It outputs **true**, if the card belongs to that family.

We include this operation in the assignment for students.

```
to family? :me :family
 op ask :me [is? :family]
end
```

The **startGame** command makes the **Bug** jump home, cleans the page and stops the "turning by **90** degrees" property of the whole **Card** family (**Card'setEvent "onClick []**).

The **go** command has to have the proper reaction of the **Bug** for the card it stands on. It has to find the name of the

```
to go
 cancel "hesitate
 let "C overlapped
 let "s ask :C [heading]
 if family? :C "Entry [step]
 if family? :C "Card1 [setHeading :s step]
```

card (**overlapped**), find the card's direction. Then the **Bug** has to set its own direction according to the direction of the card and move by the right number of steps. If the **Bug** leaves the labyrinth, it has to write the correct message and cancel the procedure. If not, it has to find out the name of the card the **Bug** stands on and set the **Bug**'s direction again. If the **Bug** stands on the **CardCross**, it has to launch the *hesitate* process.

```
  if family? :C "Card2 [ ... ]
  if family? :C "Card3 [ ... ]
  if family? :C "CardCross [step]
  if empty? overlapped
     [ifElse xSur > 330 [label "Hurray!]
                        [label "Sorry!]
      stop]
  let "C overlapped
  setHeading ask :C [heading]
  if family? :C " CardCross
     [(every 1000
         [ifElse heading = ask :C [heading]
                     [let "add 1]
                     [let "add -1]
                   repeat 90 [rt :add]]
       "hesitate)]
end
```

Semi-completed **go** command is the part of the assignment too.

Finally, we show the whole definition of the **myShape** command for the **CardCross** class.

```
to myShape
 setShape
  [pu bk 34 lt 90 pd
   setPc orange5 setFc [255 245 200]
   polygon [repeat 4 [fd 34 rt 90 fd 34]]
   pu rt 90 fd 15
   setPc "paleRed setPw 5
   pd fd 33
   lt 90 polygon [repeat 3 [fd 6 rt 120 fd 6]]
   rt 90 pu bk 48 lt 90 fd 34
   rt 90 fd 34 rt 90 fd 15
   pd fd 33
   lt 90 polygon [repeat 3 [fd 6 rt 120 fd 6]]]
end
```

The students have to solve the task in 1.5 hour. They have to demonstrate that they know how to create new classes and how to make use of their hierarchy. They have to analyze and complete partially prepared commands and develop missing events.

References

Lehotska D and Kalas I (2005), *LVF – Interface for dynamic fractions*, to appear in Proc. of Technology in Mathematics Teaching, Bristol 2005.

Kalas I (2005), *Educational software for new mathematics. About one surprising form of algorithmic skills* (in Slovak), to appear in Proc. of DidInfo, Banská Bystrica 2005.

Kalas I and Hrusecka A (2004), *The Great Big Imagine Logo Project book*, Logotron, 2004.

Hrusecka A and Kalas I (2003), *Elements of algorithmic skills for future teachers (of mathematics)* (in Slovak), proc. of DidInfo, Banská Bystrica 2003, pp. 47 – 52.