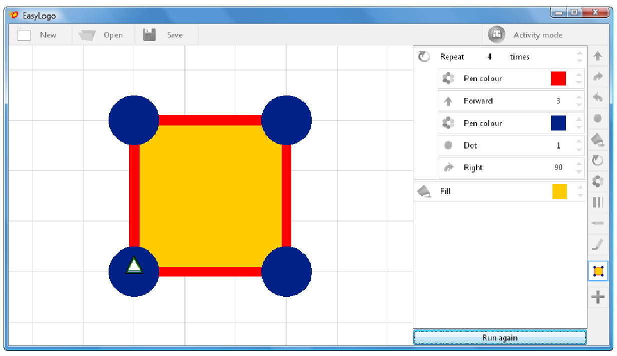
EasyLogo – discovering basic programming concepts in a constructive manner

Lubomir Salanci, *salanci@fmph.uniba.sk*
Dept of Informatics Education, Comenius University

Abstract

EasyLogo was designed for people with basic computer skills to make programming and problem solving as easy as possible for them. We wanted to use environments that would be much simpler than those in Imagine or Scratch. We, therefore, developed EasyLogo with a number of very unusual features, for example: EasyLogo works with grids in which the painting process occurs; the turtle rotates by 45°; the prog ram is constructed from simple cards, which the user then arranges with their mouse; the program is executed automatically meaning that the painting process updates whilst the user is making changes. The Logo language has been radically simplified – offering drawing commands, loops and procedures only.



```
Repeat 4 times
    Pen colour 🟥
    Forward 3
    Pen colour 🟦
    Dot 1
    Right 90
Fill 🟨
```

Figure 1. The environment of EasyLogo has an output area with a visible grid, a program editor that already contains a number of cards with commands, and icons of commands, which the user can drag into the editor. This is a so-called free programming mode where the user constructs anything that they like.

More importantly, the EasyLogo environment contains a collection of activities, which have been specifically designed to teach users basic programming concepts in a constructive manner. The consequence of such an approach is that EasyLogo blurs the boundary between computer games and programming. We believe that it is important to keep such educational environments open, in terms of activities, to allow the user to modify activities and to add their own.

Whilst testing the environment, we observed an interesting behaviour amongst users – a noticeable majority solved their assignments with loops. At first the user discovered a repeating pattern and consequently arranged commands for it; then a repeat construction was inserted and existing commands were moved into a repeat container; finally, the user changed the number of iterations for the repeat command. We revealed how very important it was that newly inserted repeat constructions had the number of iterations set to 1. For other default values, users were very confused. Such incremental processes of design, observation, analysis and improvement helped us to develop a comprehensive environment (note: this approach is close to design–based research).

Keywords

Logo, Programming, Grid turtle graphics, Didactics, Educational environment

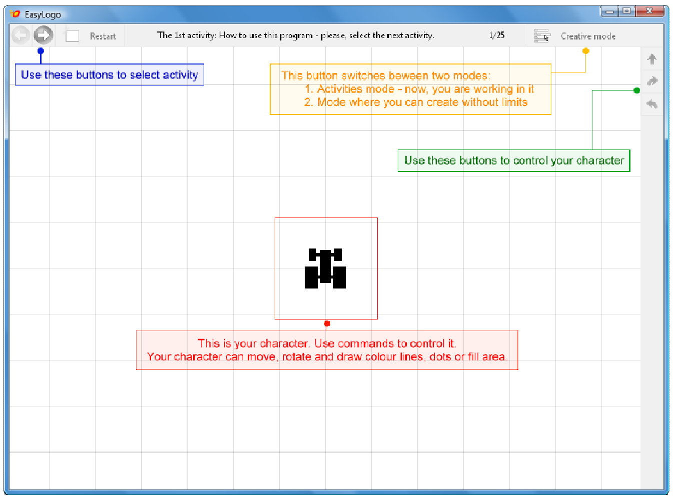
Another Logo – Why and For Whom?

A new approach to Problem-solving had to be realized for a lot of teachers at primary schools. We knew that our participants were beginners in informatics and that they would never become programmers in the future, and many of them were of an older age group with mixed levels of digital skills. Conversely, our participants were very enthusiastic and creative and, as primary school teachers, they had a vast experience of social and natural sciences. As a result, we decided to approach our project in a very constant and fun way.

We began by considering, comparing and analysing various existing environments and our findings showed problems with all of them:

- Participants found the (semi) professional programming languages extremely complicated and frustrating.
- In traditional Logo environments (such as Imagine, MSW Logo), users have to learn/memorise commands and syntax rules. This leaves room for error, such as typos or missed symbols (eg. bracket in repeat command). Ultimately, these strict conventions detered our participants.
- Alternative environments (like Scratch) are too complex with a lot of commands – building blocks.
- Children’s environments (like Thomas the Clown) are too simple and too closed in terms of their activities. So that adding new activities to them is difficult if not impossible.
- Flash games might be a lot of fun, but they were designed for other purposes with only a few of them directed at programming goals, and the majority closed in terms of adding new activities.

Despite these disadvantages, all of the mentioned environments have a lot of great features, which were a positive inspiration to us as we developed our project (see fig. 2).



The 1st activity: How to use this program - please, select the next activity.

Use these buttons to select activity

This button switches beween two modes:
1. Activities mode - now, you are working in it
2. Mode where you can create without limits

Use these buttons to control your character

This is your character. Use commands to control it.
Your character can move, rotate and draw colour lines, dots or fill area.

Figure 2. EasyLogo guides users, teaching them about how they can work within the environment. Such an approach is the norm amongst many of today’s games.

We believe that primary education should be fun, that it should use positive motivation, and that it must respect the (constructivist) theory of learning. This led us to the development of EasyLogo – a new Logo-type environment. Our principal goal was to develop an environment for primary school teachers who were studying on our course. Our secondary goal was to make the environment so simple that it could be used with older children at primary or lower secondary schools during lessons on informatics.

These were our final conclusions:

- We must simplify the original turtle graphics.
- We must keep our environment open so that any user can add new activities to it.

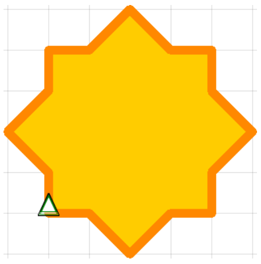
What is EasyLogo?

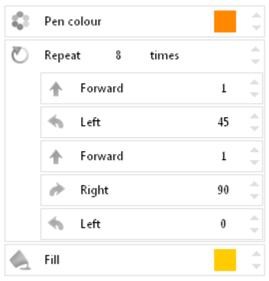
The key features of EasyLogo are:

- A simple and very intuitive user interface.
- A reduced programming construction – the environment allows loops and procedures only (no variables or object-oriented programming are implemented – for now).
- A blur between the boundary of computer games and programming – a number of activities are offered to users.
- Unusual turtle graphics – the turtle paints in a grid and rotates by 45°.

The EasyLogo language has the following commands/programming constructions only:

- `Forward` – moves the turtle a constant number of steps.
- `Left`, `Right` – rotate the turtle by 45°.
- `Repeat` – works as a container that repeats commands a constant number of times.
- `Dot` – paints dots
- `Fill` – colour fills an area.
- `Pen color`, `Pen width` – change the properties of the pen.
- `Move`, `Draw` – disable or enable drawing (like the commands, `penUp` or `penDown`).
- It is possible to create new procedures to draw shapes to specification.
- It is possible to invoke your own procedure as a command to paint a defined shape.



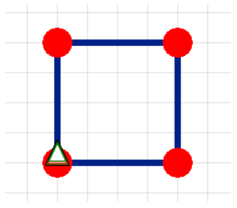


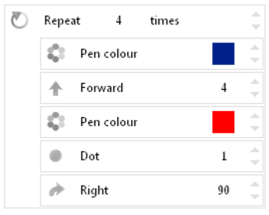
```
Pen colour              🟧
Repeat    8    times
    Forward             1
    Left                45
    Forward             1
    Right               90
    Left                0
Fill                    🟨
```

Figure 4. It is possible to draw some stars with such simplified graphics.

The `Fill` command colour fills a specified area, but it has an unusual semantic. We can see this on the previous example: firstly, we draw a closed polygon, and then we give the command to fill the previously defined area.

The `Dot` command allows the user to draw around things like: road signs, balloons or flowers.





```
Repeat    4    times
    Pen colour          🟦
    Forward             4
    Pen colour          🟥
    Dot                 1
    Right               90
```

Figure 5. Dots are the marks in the corners of the square.

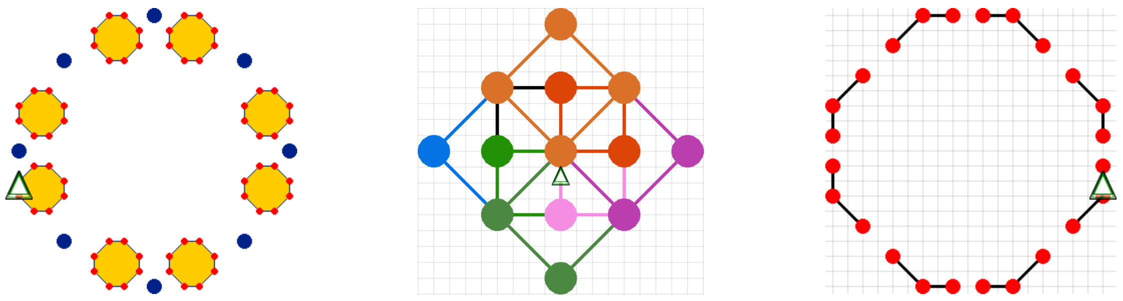


Figure 6. These are other examples of abstract pictures containing dots.

A program is constructed using cards and edited by the mouse, which means that users do not have to type commands with a keyboard. Similar approaches are used by several other environments, like Thomas the Clown, Baltie and Scratch. The precise design of the cards, alongside the mouse operations, ensures that no syntax errors are made by the user as they construct their program.

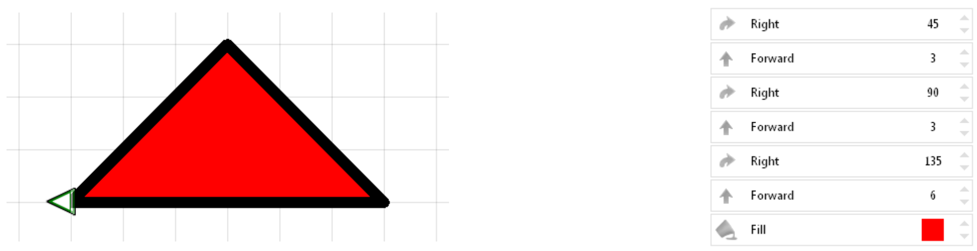
Why grid graphics?

We decided to put an accent on positive motivation from the beginning: a simple and easy to use environment, with fun activities and interesting drawings. We did not want to overwhelm our participants with complicated syntax rules, with a lot of varying commands, nor with mathematics (which, in many cases, is used improperly from a didactics point of view and complicates a beginner's understanding of basic informatics concepts). Our approach was to make the language, the turtle control commands, and the graphics as easy as possible.

We considered that it is much simpler to count steps or to calculate lengths when drawing shapes in a grid. This approach is very similar to what can be seen in some graphics editors – the grid helps users to align shapes and connect lines. Similarly, our proposed graphic pen always finishes a line at a grid point.

The 45° rotations are equally simple for calculatio ns, but are powerful enough to draw a triangle, the roof of a house, some flowers, crystals or simple stars. Thanks to grid graphics users do not need to calculate square roots...

We decided on a square grid, because a triangular (or hexagonal) one would not allow for the drawing of vertical or horizontal lines.



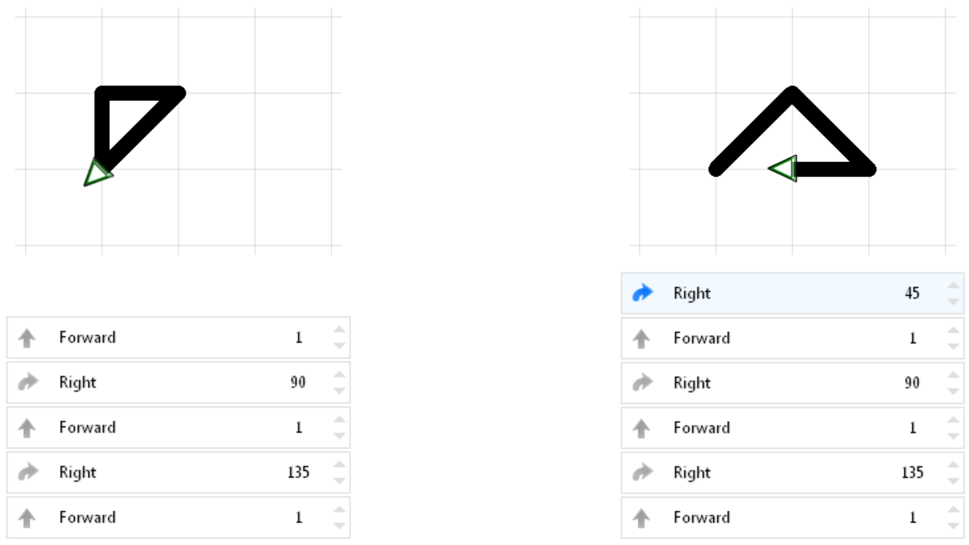
```
Right     45
Forward   3
Right     90
Forward   3
Right     135
Forward   6
Fill      🟥
```

Figure 7. This picture shows the right triangle as a roof and the commands used to draw it. Although the triangle is right, the commands contain no square root calculations.

EasyLogo generates a vector drawing as the output of a program execution, so that any resultant drawing can be freely zoomed and the user can quickly change the size (and the size of the grid) via the mouse wheel.

Weaknesses of grid graphics

The grid turtle graphic does not have Euclidean metrics (measurements of distances). This means that some shapes with diagonal lines are not immune to rotations by 45 degrees. Let's see the next example:



Forward	1
Right	90
Forward	1
Right	135
Forward	1

Right	45
Forward	1
Right	90
Forward	1
Right	135
Forward	1

Figure 8. On the left side is a small triangle. We can see what happens when we rotate it by 45°on th e right side. The original triangle shape is broken.

This is the weak spot of grid graphics. But, if we learn to accept this fact, we can still produce nice drawings or realise interesting activities.

Procedures

In EasyLogo, procedures are represented as shapes that were previously drawn.

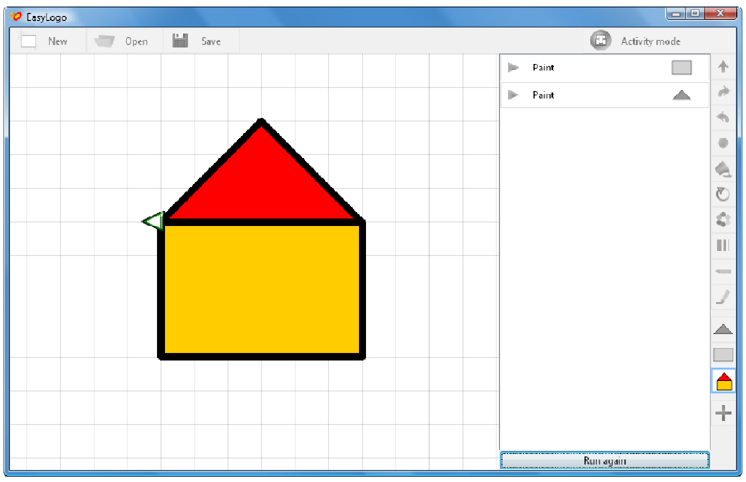


Figure 9. Picture of a house constructed from procedures – rectangle and triangle. We can see our own procedures in the right-hand corner of the window. Each procedure is represented by an icon of the shape which the procedure draws.

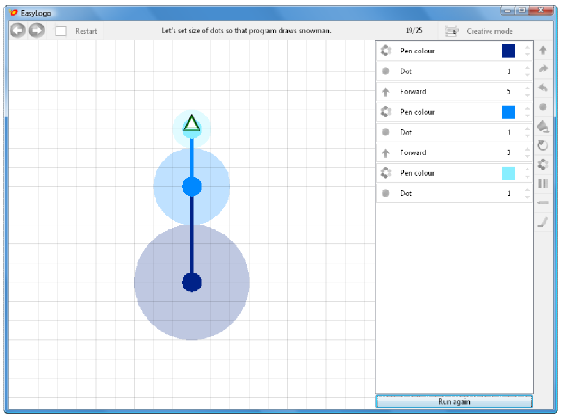
About activities

The important part of EasyLogo is the collection of activities, which are a series of small microworlds or tasks that need to be solved.

Different activities have various educational goals:

- To control the motion of a character (like a car, bee or girl) directly – by clicking on buttons.
- To control a character using a program – by constructing a sequence of commands.
- To construct a sequence of commands, which solve a simple problem.
- To recognize repeating patterns and design a loop.
- To construct procedures and use them to build a shape.
- To fix or to improve a program.

The user may arbitrarily browse, solve or skip activities. EasyLogo does not check a user's solution. A user may also switch environments from the Activity mode to the Programming mode where they can freely create any drawings that they like.

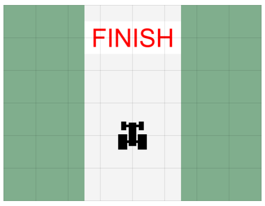


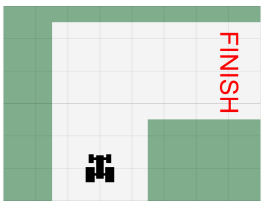
```
Pen colour
Dot        1
Forward    5
Pen colour
Dot        1
Forward    3
Pen colour
Dot        1
```

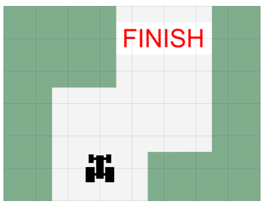
Figure 10. This activity requires the user to create a snowman by changing some parameters in the program. We can also see that EasyLogo displays a ghost image of the result below the grid.

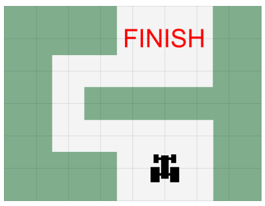
When we were thinking about the concept of EasyLogo, we thought about creating activities that varied in complexity. For example:

1. The user clicks on buttons that directly move the Car. We designed these activities in order to make the user familiar with turtle motion and rotation.







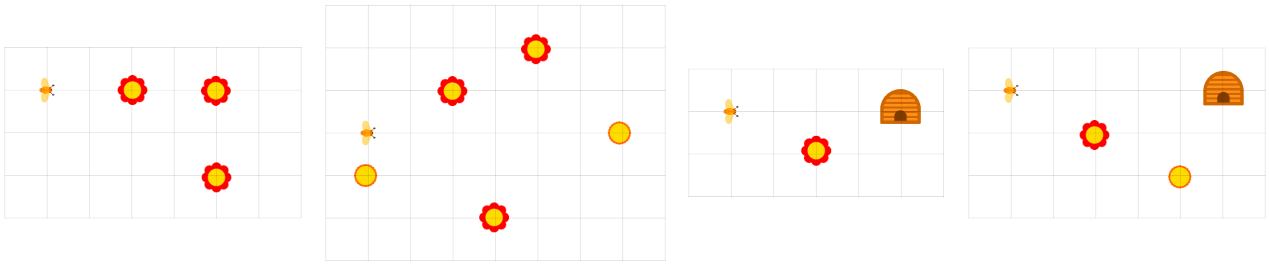


These activities are extremely simple but they are very important for developing knowledge of relative and local turtle geometry:

- The first activity could be solved just by clicking on the button.
- Consequential activities require a combination of rotations and movement .

The difficulty of each activity is defined by a road that the car must navigate, especially by the number of bends in each road.

2. While the previous series of activities required just clicking on buttons, the following series requires programming. These activities teach the user to construct a linear sequence of commands from `Forward`, `Left` or `Right`. We used the idea of a Bee that needs help to visit a number of flowers.

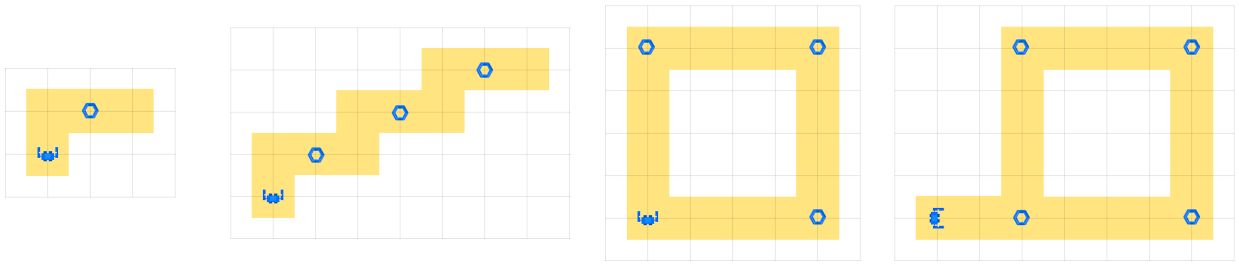


The activities are graded in levels of complexity:

- The bee must visit 3 flowers (in any order),
- The bee must visit more flowers (in any order),
- The bee must visit all of flowers and then fly into the beehive.

There are several skills that the user needs to master here: how to imagine a path for the Bee (which is also a very simple graph problem), how to drag and to order commands, or how to discover and change parameters for some of the commands.

3. Activities for `Repeat` commands were designed in order to teach the user to recognise repeating patterns.



In several activities, we used the idea of a robot that needs to collect messy gadgets:

- The solution to the first problem will be the body of the future loop.
- The user should (re)discover a repeating pattern and construct a loop with a repeat construction.
- In the more complex problem, the robot should realize something different before the loop itself (see the last picture in the series).

There are additional activities for loops that require the user to solve more complex problems. For example, they have a more complicated body of loops or they need to combine loops with procedure calls etc.

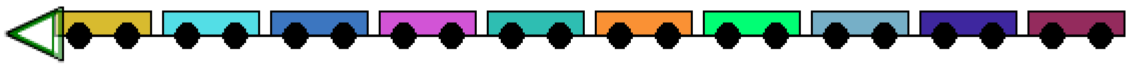
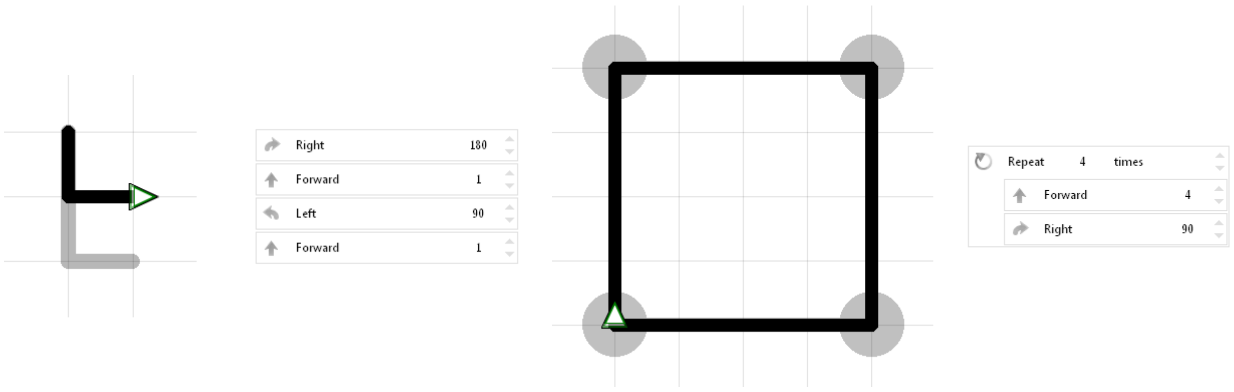


Figure 11. Example of a more complex activity: the train is constructed from several wagons.

In this way we could continue and describe educational goals and analyse our didactics approaches, for example, with other activities. But we then realized that we could use the existing scheme of activities, but organize them according to didactics principles:

- We always start with a very simple and elementary problem.
- Then we allow users to gain in (practical) experience.
- And only then do we introduce them to a slightly more complicated situation.



```
Right    180
Forward  1
Left     90
Forward  1
```

```
Repeat  4  times
    Forward  4
    Right    90
```

Figure 12. These are examples of activities in which the user has to fix programs – for example, to draw the letter L or to draw a square with dots. Gray drawings that are displayed behind black parts are requested results.

To create a graded series of activities we had to recognize and to qualify various levels of complexity. According to our empirical experience, this is the hardest problem for many people in didactics and the teaching of programming.

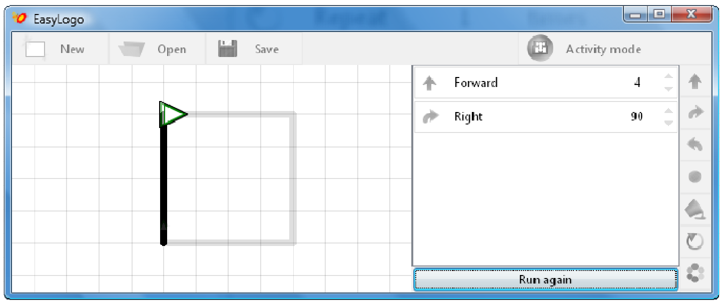
Interesting facts, observation, results

1. We believe that it is important to keep educational environments open in terms of activities. EasyLogo was designed in such a way that anyone can modify or create new activities. We decided to use simple textual files as a description of each activity and its properties. Optionally, each activity may display a bitmap as a background drawing.

2. We developed EasyLogo in order to draw pictures immediately as the user made changes to the program. This means that EasyLogo always executes a program automatically as the user inserts/removes commands or sets parameters.

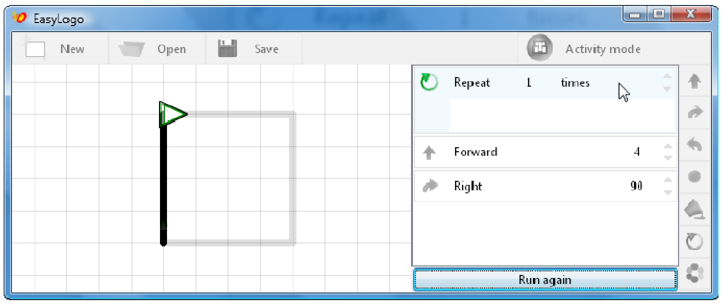
This was to become an incredibly useful feature, because it gives immediate visual feedback during the construction of a program. Sometimes, it is fun, interesting and edifying to see changes at once as we “play” with programs or parameters.

3. Thanks to the previous feature, we observed how people construct algorithms with loops. We will illustrate this process using the example of drawing a square – let’s have a look at the following images and comments below them:



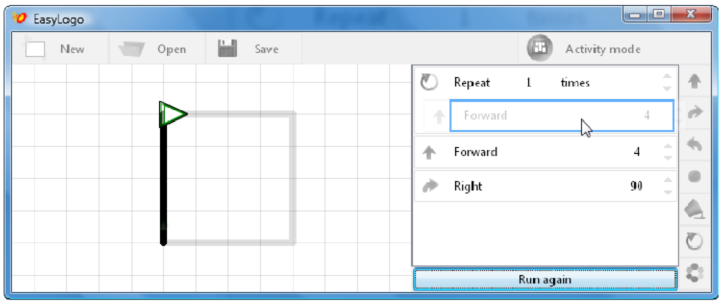
Forward 4
Right 90

People find the repeating pattern first. Then they arrange commands which will become the body of the loop:



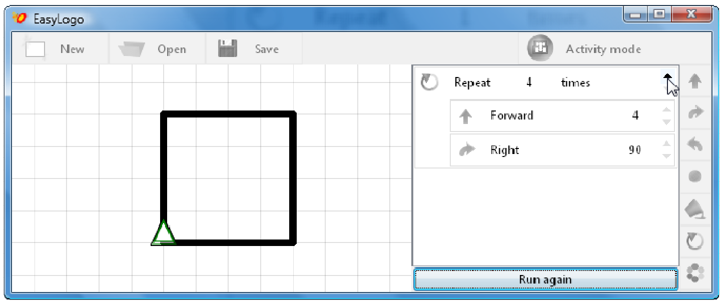
Repeat 1 times
Forward 4
Right 90

People insert the repeat command later. They usually put this command at the first position. Please note that, by default, the repeat command has set the number of iterations set to 1.



Repeat 1 times
Forward 4
Forward 4
Right 90

Then, they move previously arranged commands into the body of the repeat construction.



Repeat 4 times
Forward 4
Right 90

Finally, they change the number of iterations.

We saw this method widely used amongst young children as well as many adults.

4. There is an interesting story as to why the default number of iterations is set to `1` for the repeat command. In the early version of EasyLogo the repeat command had the default number of iterations set to `4` so that a newly dragged command said “`Repeat 4 times`”. But we had to change the value from `4` to `1` in later versions.

Our colleague Monika Tomcsanyiova has a `6` year old son, Tomas, with whom she decided to test EasyLogo. She observed that Tomas worked exactly in the same way as we have just described. However, Tomas became very confused when he started to move commands into the body of the repeat.

As we discovered, he was confused because all of the moved commands were automatically executed `4` times, whereas others were executed only once. As a result he saw some strange drawings appear whilst he was working and this did not make sense to him. The program drew something other than what Tomas had expected, and this is reason why the repeat command now has its parameter set to `1` by default.

This experience shows that we must carefully consider every detail of an educational environment when using programming languages with beginners. Otherwise, the resultant product is counterproductive rather than useful.

5. We showed EasyLogo to another group of teachers; those who are teaching programming at lower or upper secondary schools. EasyLogo was introduced to them as an example of a programming environment that was primarily developed for educational purposes in contrast to Java, C# or other languages or environments designed chiefly for the software industry. The teachers were very positive and were absorbed with EasyLogo when using the program. To be correct, however, we must say that we do not, as yet, have feedback from schools where children have used EasyLogo – simply because this environment is too new.

Conclusion

Our goal was to introduce programming to beginners in an easy to understand way by designing an environment which blurs the boundary between computer games and programming. However, we still believed in keeping to fundamental informatics goals, such as: problem solving, the designing of algorithms, the use of programming constructions, the decomposition of problems into smaller parts, how to solve simple graph problems or how to work within a grid structure. For this purpose we designed a series of activities, graded according to their complexity in problem solving, to help users steadily develop their skills and knowledge.

We believe that EasyLogo combines several unique features:

- We implemented and analysed turtle graphics in a grid.
- We designed the fill command.
- We created activities that were an intrinsic part of the environment itself.

Technically, a user constructs a syntax tree by dragging commands using a mouse. The display list is generated from such a structure and is then rendered on the screen.

Our incremental process of design, observation, analysis and improvement is close to design–based research. For example, this approach helped us to make the repeat construction far more intuitive and comprehensible for users.

References

Kalas, I. and Blaho, A. (2001) *Object Metaphore Helps Create Simple Logo Projects*. Eurologo 2001, 55−66.

Correeia, T.and Correeia,S. (2007) Natural *Visual Programming Languages*. Eurologo 2007, 44.