

Imagine... a new generation of Logo: programmable pictures

Ivan Kalas & Andrej Blaho
Comenius University
Bratislava, Slovak Republic
E-mail: kalas@fmph.uniba.sk

Abstract

In 2000 we completed the development of a new generation of Logo environments containing a radical combination of the direct manipulation interface and rich interactive programming language. The “Imagine” environment will be released by Pearson Education, see <http://www.logo.com>.

In [1] we presented a brief overview of the conception of “Imagine” from the viewpoint of the user of SuperLogo. In this paper we characterise in depth one of its innovative concepts – programmable pictures, that is, pictures (shapes of turtles) specified in the Logo language itself. We give a brief overview of the history of the concept of shapes of Logo turtles. We present a sequence of topics to illustrate the power of this concept for educational applications: both for beginners and advanced users. We argue that “Imagine” provides a tool for facilitating the learning process.

Keywords

Imagine, Logo, programmable pictures

Introduction

In [1] we introduced “Imagine” as a new generation of Logo, with object-oriented structure merged into traditional Logo philosophy, with empowered animation, open hierarchy of graphic screens and panes, hierarchy of objects and behaviours, parallel independent processes, direct painting tools, extended direct manipulation interface, tools for publishing for the Web, rich Logo language and other characteristics. When developing this educational environment we had in mind a learner who wants to access a very broad palette of activities, from painting and animating to Web authoring, doing traditional Logo, creating multimedia, using speech input and output, modelling, constructing domain-specific learning frameworks, communicating ideas, building presentations, developing projects and microworlds for numeracy, literacy or science, working with data... Our goal has been to provide students, teachers and developers with a challenging general tool for learning.

When we started developing SuperLogo at the beginning of the 1990s, see [2, 6], we decided to retain as a principle the original philosophy of Logo. At that time we decided to develop Logo within the Environment for Environments Metaphor, see [2]. That is, in harmony with [3] and [4] we tried to empower the Logo environment so that it could serve children, students, teachers, and the developers of educational domain-specific applications. We decided to make Logo a strong tool for learning, programming and developing all in one.

Shapes of turtles

Probably the most important “discovery” during our development of SuperLogo was the approach to the shapes of turtles, see [5]. We managed to find very natural, intuitive, and yet powerful solutions that opened broad new fields of topics for users' projects and activities. Let us briefly overview the history of turtles' shapes in older versions of Logo.

Initially, the shape was an outline of a small turtle or a triangle, which had several rotations, built in the core to display the actual heading of the turtle.

Later the Logo community realised that it was not always the drawing on the screen itself which was the real result of doing Logo. Sometimes it could be a kind of composition of Logo drawing **and** the turtle itself. Then, however, it would be more appropriate if the turtle looked like a ship, an animal, a rocket etc., or even better: several ships, animals, rockets... Thus, a fixed number of turtles (4 or 16 etc.) could have been involved, together with the possibility of choosing their shapes from a fixed library of black-and-white shapes of fixed size. Those shapes, however, did not display the actual heading in any way.

From there the step was to offer a kind of **shapes' editor** to open a shape from the library and either modify it or clear it and create a new one. Then the restriction of the fixed size was removed.

Newer versions have become more powerful and attractive by allowing the user to set any picture as a turtle's shape. New pictures have either been **loaded from a file** or **taken as a picture** from the screen (which made it possible to draw them in a Logo way by the turtle, then take a snapshot of them) and set the result to the turtle as its new shape.

In SuperLogo we have introduced a new data object called **image**, see [5]. The decision to work with images in exactly the same way as with words and lists proved to be very powerful: similar to words consisting of characters, images consist of **frames**. An image is a sequence of frames, which can be interpreted by the turtle in two different ways:

- For a traditional turtle, frames of its shape serve to display its actual heading. There is an automatic mechanism in SuperLogo to choose and display the corresponding frame for the actual heading.
- If the user declares a turtle to be an **animation one**, the connection between **frames** and **actual heading** is broken and it is up to the user to decide when the turtle will display which frame. In this way it is possible to create the illusion of a movement, see Figure 1.

“Imagine” has made a step in two different directions:

1. We decided to remove the distinction between *traditional* and *animation turtles*. We generalised the **concept of image** in the following way: each image is a sequence of **frames**, which are interpreted as different pictures for displaying different headings of turtles. Each frame may consist of one or several **frame items**, which perform like an animated GIF. If the frame consists of one frame item, the shape is static. If it consists of several items, “Imagine” will automatically alternate them. Thus the engine **automatically** creates and controls the illusion of animation.
2. The second direction is also quite pioneering. To define the shape of a turtle, **you can use the Logo language itself**, see below.



Figure 1: Frame of SuperLogo

Introducing animation turtles into SuperLogo has increased its power because it made it easy to create animated compositions and stories, where the goal is **the process itself**. In spite of that, we have always felt that there have been two disadvantages in connection

with animations. The first of these has disappeared by moving from SuperLogo to “Imagine”, the other has not:

1. The user had to define his/her own main animation control loop. This is not needed in “Imagine” because an independent process does this. The generalised structure of images shifted the task of controlling animations from user to the engine itself.
2. To place the animated turtles into your composition requires that you either create corresponding images or get them from somewhere else. Both ways are difficult and too demanding; users are forced to rely on libraries of professional ready-made images.

Pictures described by Logo language

When working with children we have often felt that there is a wide gap between introductory Logo activities and any kind of possible continuation – a kind of threshold which prevents many young users (and their teachers!) from doing the second step. In [6] we illustrated how to overcome it and fill this gap with using Logo data structures in challenging combinations with turtle geometry.

In “Imagine” we decided to try yet another approach. When Logo beginners learn how to type in a sequence of instructions, or define simple drawing procedures (like **house**, **tree**, **boy...**), let us show them a way to create new **shapes for turtles** using Logo instructions and simple procedures for drawing. At the moment they are able to draw a square (for example in a thick red pen) by saying:

```
? setPC "red setPW 8
? repeat 4 [fd 60 rt 90]
```

They can use the same piece of program to specify a square shape for the turtle. They simply enclose their piece of program into square brackets (thus creating the drawing list) and use it as the input to **setShape**:

```
? setShape
[setPC "red setPW 8 repeat 4 [fd 60 rt 90]]
```

The square will become the turtle's new shape. It will not be drawn in the background picture, it is a turtle that lives **above the background**. What makes the difference between the two squares created above?

1. The second square is a turtle; that is, it is prepared **to be moved** by simple turtle commands **fd** and **bk**. It can also be moved by the **direct manipulation tools**. The teacher or students can define a class of turtles (for example, **myTurtle**)

with only one setting of the `Turtle` modified: let the value of `autoDrag` be set to `true`. Thus, each turtle created as an instance of `myTurtle` class can be dragged by the mouse along the screen with no extra programming work:

```
newClass "Turtle "myTurtle [autoDrag true]
```

- If the turtle's shape is specified by a **drawing list**, it can **automatically be rotated**. If you say `rt 10` it will react quite intuitively – the shape will be redrawn in a new heading. If, for example, it is a `tree`, you can easily make it bend from one side to another by `lt` and `rt` commands.

Topics based on automatic rotations

Let us develop a procedure to draw a sail of the traditional Dutch windmill. It may be:

```
to propeller
setPW 2 setFC "taupe5
repeat 4
[fd 53 bk 43 polygon
[repeat 2 [fd 40 rt 90
fd 10 rt 90]]
bk 10 rt 90]
end
```



Figure 2: Windmill

Click the **New Turtle** button of the **Main Bar**, then click in the graphics screen, thus creating the second turtle (named by default `t2`). Let its shape be similar to the windmill house that supports the sail, see Figure 2. We may now combine this pair of turtles and make the sail rotate. The bottom part will be static (`t2`); the rotating part (`t1`) will get its shape by inserting `setShape` command into `sail` command. Let us make the sail rotate by saying `repeat 360 [rt 1 wait 20]`. Such a solution, however, has two drawbacks: (1) the sail will rotate only for a while, then stop (2) while it is rotating, the command line will be blocked.

Let us therefore use another approach: we will define an **endless parallel process** for the sail `t1`. It will independently keep the sail rotating forever:

```
? t1'forever [rt 1 wait 20]
```

This instruction launches the parallel process, then terminates. Thus the command line is blocked for a very short period only.



Figure 3: Compositions of turtles – pieces

Compositions of several turtles

Many creative activities are based on building compositions of small coloured pieces of different basic geometrical shapes – like squares, sticks, circles and half circles, various triangles etc. The goal of such activity is to build objects like flowers, cars etc.; to find basic geometrical shapes in a real picture displayed as a background of the page and cover them by the pieces; to complete partly constructed scenery by adding other pieces; to create geometrical compositions with repeating patterns... and many others.

In such microworlds a palette of pieces is usually provided, from which you can choose and take a copy of a piece (from an endless supply). In *"Imagine"* we can implement such microworlds, for example, by defining a simple class of objects named `Piece`. An instance of this class will be a turtle with its pen up, with the `autoDrag` attribute set to `true` and with its shape specified by a simple piece of Logo program – a **drawing list** – chosen from a list of alternatives:

```
[setFC "blue polygon [4 [50 90]]]
[setFC "yellow filledCircle 30]
[setFC "red polygon [2 [50 90 100 90]]]
[setFC "red
polygon [0 -90 3 [25 120 25 0]]]
```

These are drawing lists corresponding to pieces in Figure 3. What are the advantages of this approach?

- It is easy to modify the list of accepted shapes.
- The problem with multiple layers of objects in the composition turns trivial.
- Dragging all pieces-turtles is trivial.
- It is easy to create a copy of a piece, for example, by using the `clone` command.
- It is easy to rotate any piece by any angle (simply by `right` and `left` commands).

Planar geometry

Another strong motivation for us to offer **drawing lists** in *"Imagine"* has come from planar geometry, where basic elements are circles, lines and points. A short drawing list can easily define any of them. For example, a line can be realised as a long segment:
`t1'setShape [fd 1000 pu bk 2000 pd fd 1000]`

Let us define the new class `Circle` as a sub-class of `Turtle`; each instance will have its pen up, will have a default shape of a circle with diameter `40` and a red point on its circumference. Next to the point there will be the name of the object (see `onCreate` event):

```
? newClass "Turtle" "Circle
  [pen pu font [Fixedsys [10 400 0 0 0
238]]
  heading 90 shown false Radius 20]
```

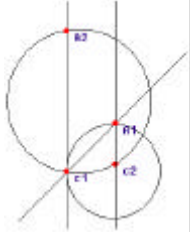


Figure 4: Turtles as circles and lines

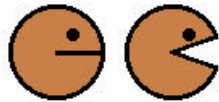


Figure 5: Two drawing lists

Let us define three events for `Circle`, namely `onCreate`, `onLeftDown` and `onDrag`. The `onCreate` event inserts the name of a new object into the label part of the drawing list. The `onLeftDown` event analyses whether you clicked a black circumference of a circle or its “handle” – a small red point. Based on the result, the `onDrag` event will either **drag** the whole circle (if you clicked the black circumference) or **resize** the circle otherwise.

Now each call of `new "Circle [...]` will create a new circle as an instance of the class `Circle`. Similar to `Circle` we can define the `Point` and `Line` classes and further develop this microworld, so that each object is aware of other objects in the plane.

Animated programmable pictures

Another powerful extension liberates the user from technical details. In “Imagine” you can use an image with several frames to create an animated actor without any additional effort – the engine itself will control the animation. Similar mechanisms work also for the drawing lists: if you set a **list of several drawing lists** to a turtle as its shape, the items will automatically be alternated. Thus, an illusion of animation based on a sequence of programmable pictures will be created. Figure 5 illustrates how two drawing lists can represent a figure of Pacman. In this way, the overall problem to be solved is divided into several well defined independent sub-problems.

Drawing lists for developers

Drawing lists prove to be a strong tool, because they allow very complex shapes, polygons or splines to be created without considerable demands on memory space. Note that the language used in drawing lists consists of a powerful, yet restricted, subset of Logo. Drawing list may, for example, contain a text specified by the `label` command. If we use a **vector**

font, the text will rotate together with the picture. Sometimes, however, we can on purpose choose a non-vector font. This may sound reasonable if we have to create a library of geometric shapes, points, circles etc., see Figure 4.

Note also that if an item of a drawing list is a word prefixed by a colon `:` (a variable) or if it is enclosed in brackets `(...)` (an expression to be run), it will be the first we evaluated by “Imagine”. Only then will the drawing list be built and use procedures as their input. In that way the actual appearance of the drawing list may be continually recomputed by a running process.

Conclusion

There are several other ways in “Imagine” to exploit **drawing lists**. We believe that the idea of specifying the shape of an a turtle through Logo language extends the applicability of our tool for developing complex projects with less effort spent on solving technical questions. We believe that features like **programmable pictures** make “Imagine” an appealing powerful educational software environment.

Reference

1. Blaho, A., Kalas, I. & Tomcsanyi, P. (1999) *Open Logo – A New Implementation Of Logo. Proceedings of EuroLogo 99*, Sofia.
2. Blaho, A., Kalas, I. & Matusova, M. (1994) Environment For Environments: A New Metaphore For Logo. In Wright, J. & Benzie, D. (eds.) *Exploring A New Partnership: Children, Teachers And Technology*. IFIP North-Holland.
3. Turcsanyi-Szabo, M. (2000) *Subject Oriented Microworld Extendible Environment For Learning and Tailoring Educational Tools*.
4. Sendova, E. & Ivanov, I. (2000) *Lifting the hood to see how something works*. ICEUT-2000.
5. Blaho, A. & Kalas, I. (1995) Playing, Developing and Computing With Images in Comenius Logo. *Proceedings of EuroLogo 95*, Birmingham.
6. Blaho, A. & Kalas, I. (1997) I Beg Your Pardon Turtles: Don't Forget About Data Structures. *Proceedings of EuroLogo 97*, Budapest.

Biography

Andrej Blaho and Ivan Kalas are senior lecturers at Comenius University, Bratislava. They are the authors of SuperLogo.