

# Magic Imagine

**Károly Farkas**, [farkas.karoly@nik.bmf.hu](mailto:farkas.karoly@nik.bmf.hu)  
John von Neumann Faculty of Informatics, Budapest Tech

**Ildikó Tasnádi**, [tasnadi\\_ildiko@hotmail.com](mailto:tasnadi_ildiko@hotmail.com)  
Informatics teacher, Budapest

## Abstract

Imagine is a new generation of the Logo-languages, which combines traditional Logo-technique with an object-oriented approach, in line with today's programming trends, providing an ideal environment for the playful method of teaching IT. At the same time, it offers a more sophisticated programming platform for advanced students, as well.

We can begin the playful introduction of the basics of object-oriented programming to children already experienced in the use of Logo-language. It is enough to start with some basic notions, such as: class, entity, parent, descendant. Explanations can be derived from their micro-worlds, using their vocabulary and lots of illustrative examples. Succession, for instance, can be demonstrated by means of family relationships, highlighting similar (inherited) and different (unique) features of the parent and the descendant. This would then be followed by abstraction of all of the above, starting from the specific example to then reach the general concept.

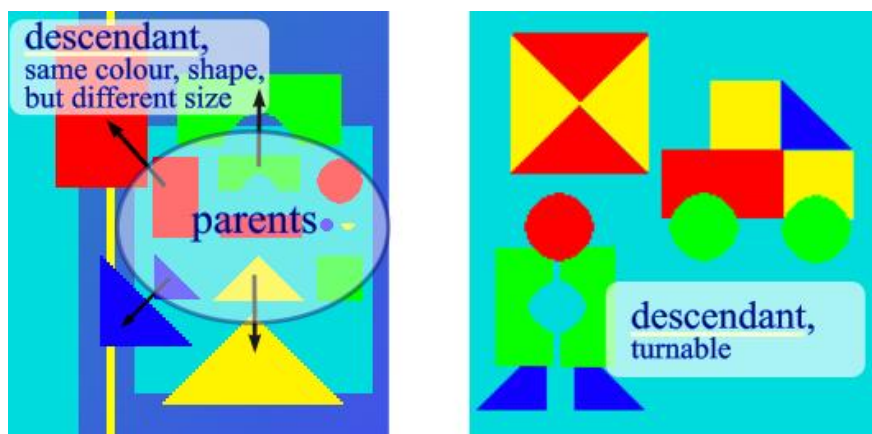


Figure 1. Similarities and differences between the parent and the descendant

Imagine has several **pre-defined primitive classes**, based on which we can easily define our own individual subclasses and instances.

First of all, we have to pre-define our expectations towards our class, in terms of the properties (conditions) and events (procedures) it should possess.

The class determines the behaviour of each turtle of this kind in various situations. Contrary to normal turtles, though it is invisible and it exists in the background only. Only after this can we create the specific, already visible entities.

## Keywords

Imagine, object-oriented approach, object-oriented programming, playful IT – teaching

## Recent possibilities in Logo: object-oriented programming playfully

Once the abstractions have been adequately prepared, by learning a few commands and expressions we can basically start our object-oriented program. To facilitate the understanding of these abstractions, situations need to be devised, which pupils can identify with.

To illustrate this exercise, we created a program called "BUILDING BLOCKS". In this program there are building blocks on the right side of the screen (these are the parent classes), from which we can choose by clicking on them, creating a copy that can be dragged (the descendant) which can be used for building.



Figure 2. The program

In the learning phase, classes should be generated using only already existing pre-defined classes (e.g. Turtle).

We can define a new class using the newClass command in the command-line.

```
? newClass "parent_name "myClass_name [setting-list]
```

Example:

```
? newClass "Turtle "ParentBuildingBlock []
```

This new turtle-class is invisible on the screen, but with the Project Manager (F4) it can be made "real and tangible" for children, as well.

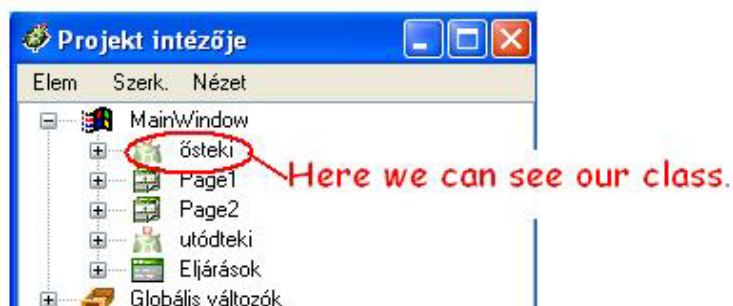


Figure 3. Our class is "real and tangible" for children.

Using the Change Me dialogue we can set and modify different settings in a quick and easy way, thus avoiding the command line.

We would like our new class to have the following properties:

- If we *press the right mouse-button*, the turtle should change its colour (red, blue, green, yellow). This is easiest done using frames in LogoMotion.
- If we *press the left mouse-button* then a new building block with the same colour and shape should be created, in the same position, but of double the size of the basic element (width, length \* 2), and it should of course be draggable.

Implementation:

Open the Change Me dialogue of the ParentBuildingBlock.

1. Event → Add → onRightDown: `setframe frame + 1`

2. Event → Add → onLeftDown:

```
new "cube [pos (pos) shape (shape) frame (frame)
          shapescale (shapescale * 2) .name "block]
ask lastname [startdrag]
```

To ensure selection from among the colours, a command should be entered into the command line:

```
? setframemode "true
```

Prior to the development of the specific individual units, the turtle class called Cube should be defined, specifying the properties of the descendants.

```
? newClass "Turtle "Cube [autodrag true pen pu]
```

Later on the properties of the descendant turtle named Cube can be modified, as well, using the Change Me dialogue. (At this point it should be repeated, that the descendant class may have properties different from those of the parent class, in addition to the inherited properties, since offsprings may have features different from their parents.)

Open the Change Me dialogue of the Cube.

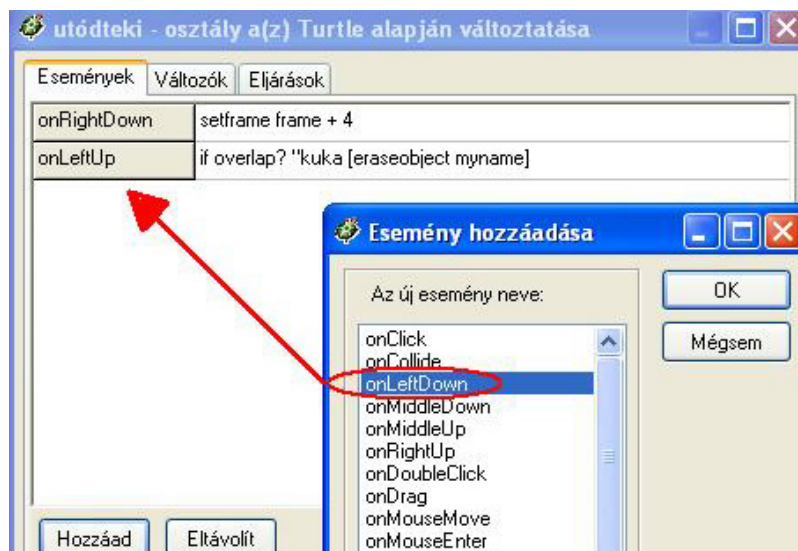


Figure 4. Change Me dialogue

Event → Add → onLeftDown : `tofront`

Subsequently, the turtles visible on screen, as well, are generated.

```
? new "BuildingBlock
```

A new turtle bearing the pre-difined BuildingBlock features appears on the screen. All there is left to do, is change its shape, to arrive at the first building block element. It can be given a new name, buildingblock1, for instance.

A few more turtles of different shapes are generated in the same way and we have the basic building blocks (e.g.: buildingblock1 – buildingblock10).

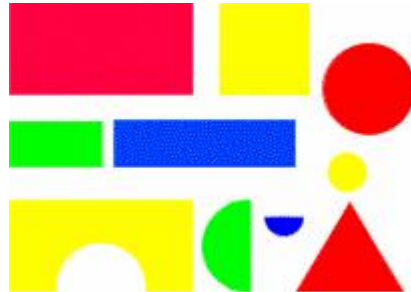


Figure 5. buildingblock1 – buildingblock10

A rectangular area should be defined as a drawing surface and the possibility to delete should be provided.

There are two ways for deletion: elements no longer useful can either be thrown into a waste bin or the entire drawing sheet can be wiped clear pressing a single button. For the one-by-one deletion of elements we need a simple turtle which can wear the shape of the bin and an event (onLeftUp) is added to the properties of the Cube class. Let us assign to it the following:

```
if overlap? "bin [erase_from_list eraseobject myname]
```

*Erase\_from\_list* is a procedure which can be written in Cube's Change Me dialogue.

```
to erase_from_list
  make "members butmember myname :members
end
```

In order to be able to erase from a list, we first need to create it. The easiest way to do that is to add it to the list when it is created. Thus extend ParentBuildigBlock's onLeftDown event to include a procedure that performs this operation.

```
new "cube[pos (pos) shape (shape) frame (frame)
  shapyscale(shapyscale * 2) .name "block]
ask lastname [startdrag put_into_list]

to put_into_list
  make "members lput lastname :members
end
```

But, first of all, a command should be entered into the command line:

```
? make "members []
```

The building we have thus created can be deleted in one step, as well, if the following procedure is entered to the onPush event of a button (X):

```
to erase_list
  if not empty? :members
  [
    make "effaceable first :members
    eraseobject :effaceable
    make "members butmember :effaceable :members
    erase_list
  ]
end
```

The result of this command is that all of our turtles bearing the name "blockn (all descendants) will be deleted from the memory and will thus cease to exist.

At the end, create a close button and an event (onPush) is added to the properties of it:

```
(bye "false).
```

Ideas for further development:

Let us modify the program in such a way that the descendant turtles can be rotated (by 30, 45, or 90 degrees) by clicking on them.

## Summary

The future belongs to object-orientated programming. The earlier children become acquainted with this approach, the easier it will be for them to use it in their work. We can start teaching it already in primary school when children are open and ready to learn new things and ideas. In our paper we have attempted to illustrate this through a simple example.

## References

A. Blaho, I. Kalas, L. Salanci, P. Tomcsanyi (2001) *Imagine Reference Guide*, Logotron