

Educational Technology Lab

National and Kapodistrian University of Athens

School of Philosophy

Department of Educational Studies



Director: Prof. Chronis Kynigos

“MaLT2” Manual

```
TO cube :x
repeat 4[
square :x fd :x down 90 ]
END
rt 90
cube 60
pd
pu
dn 90
up 90
up 100
pyramid 80 30
home
ladder 20 50 90 3
```

τετράγωνο ορίστηκε.
μετατόπισε κέντρο ορίστηκε.
πυραμίδα ορίστηκε.
τετράγωνο ορίστηκε.
κύβος ορίστηκε.

Μεταβολές της συνάρτησης πυραμίδα

Όνομα Από	Έως Βήμα
x 43	87 174 1
w 15	30 60 1

TABLE OF CONTENTS

1	MaLT2	4
1.1	<i>What is MaLT2</i>	4
1.2	<i>Description of MaLT2</i>	4
1.2.1	<i>The 3D scene and the avatar</i>	4
1.2.2	<i>The Editor</i>	5
1.2.3	<i>The Variation tools</i>	5
2	Features of the 3D scene	6
2.1	<i>Camera controls</i>	6
2.2	<i>Scene's toolbar</i>	6
3	Basic control – avatar guidance	7
3.1	<i>What is a command?</i>	7
3.2	<i>Movement of the avatar on the surface</i>	7
3.3	<i>Calculations using MaLT2</i>	9
3.4	<i>The avatar's trace</i>	10
4	Structural language features in MaLT2	11
4.1	<i>Primitives</i>	11
4.2	<i>New commands (Procedures)</i>	11
4.3	<i>Procedure construction</i>	12
4.3.1	<i>Procedure inputs</i>	13
4.4	<i>Sub-procedures & Hyper-procedures</i>	14
5	Dynamic manipulation – Variation tools	16
5.1	<i>The Variation Tool</i>	16
5.2	<i>The 2D Variation Tool</i>	17
5.3	<i>An example</i>	19
6	Repetition structure	21
7	Recursive procedures	23
8	Conditional commands	25
8.1	<i>Control commands in recursion</i>	25
9	Other MaLT2 features	26
9.1	<i>Save files</i>	26
9.2	<i>Open files</i>	27
9.3	<i>Other features</i>	27

Appendix A – Tables of Commands	29
<i>Table 1: Avatar control commands</i>	29
<i>Table 2: Programming structures</i>	32
<i>Table 3: Mathematical commands</i>	33

1 MaLT2

1.1 What is MaLT2

MaLT2 (MachineLab Turtleworlds 2) is an online tool of symbolic expression in mathematical activity by means of programming for the creation and tinkering of 3D dynamic graphical models. The MaLT2 website is <http://etl.ppp.uoa.gr/malt2/> and it can be accessed from any device with internet access, without requiring any additional installation.

1.2 Description of MaLT2

MaLT2 consists of three distinct and yet connected work areas. These areas are called components. As shown in figure 1, the three components are the '3D scene', the editor and the variation tools as they are being described below.

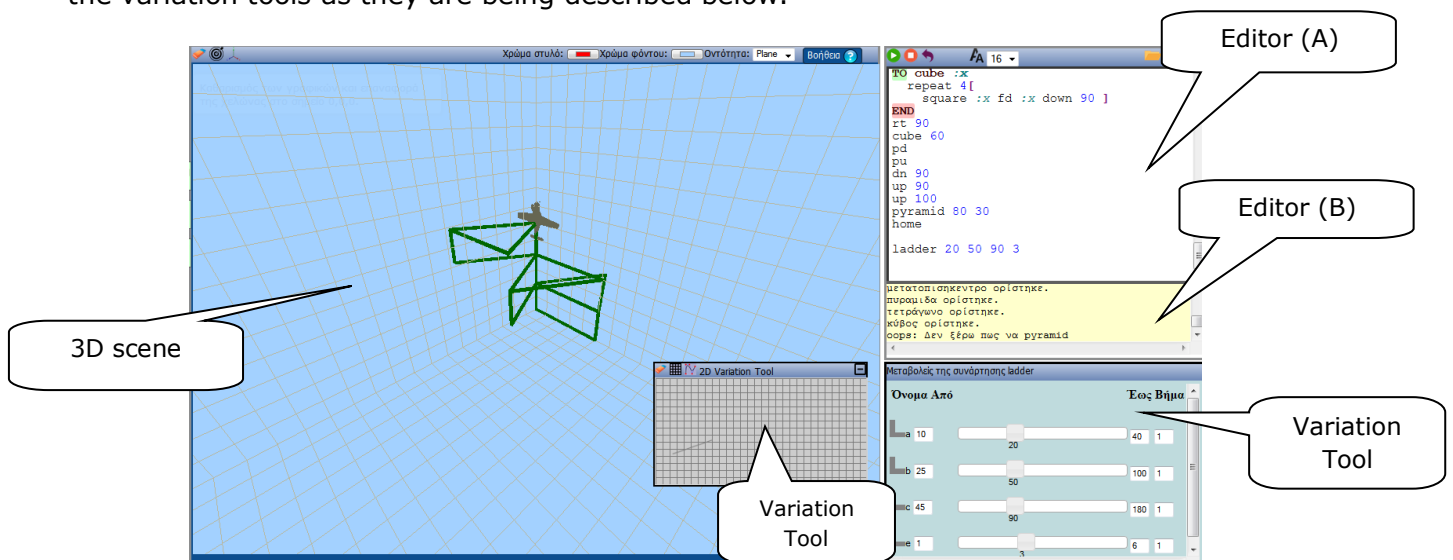


Figure 1: Screenshot of the MaLT2 environment

1.2.1 The 3D scene and the avatar

On the left side of MaLT2 appears the component of the '3D scene', which also includes the avatar. The avatar is a 3D object which you can move in the 3D space by executing some Logo commands. The initial 3D object of the avatar is a humming bird, but you may change it from a dropdown menu at the top toolbar of the scene. The 3D scene is a surface on which the avatar leaves a trace as it moves (unless you choose that it does not leave a trace). The scene also contains a 3D grid to help you orientate in the 3D space. The three axis (**xyz**) that are used in the scene's 3D area are shown in figure 2 below:

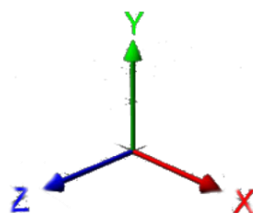


Figure 2: The three axes of 3D scene

The imaginary shape of the scene is a big invisible sphere. Thus, when the avatar moves on the scene, it follows the sphere shape.


The scene's avatar, apart from the 3D object that represents (bird), it is also defined by its state, in other words by: a) its position and b) its orientation. Its position is determined by the center of the circular object and its orientation is defined by the position of its head. As the avatar moves, it leaves behind a 3D trace, like a small cylinder. This trace has a specific trace color (red by default), which you can change by the top toolbar of the scene. Finally, you are able to look at the scene from different points of view through the rotation of the camera or through "zoom in / zoom out". These tools are placed at the left bottom corner of the scene.

1.2.2 The Editor

In the area of the 'Editor' command component you may a) write whatever you want, i.e., text, numbers, arithmetic calculations in the same way we use a word processor and b) write commands and programs that enable the avatar to change its state. This is realized with the use of a programming language called Logo, that derives from the ancient Greek word 'Logismos' (Calculus). Logo language contains a series of commands and constitutes an easy way to define your own commands; as many as you wish. The commands control and guide the scene's avatar and define the changing values of its state attributes; position and orientation. Each time a command is executed, the avatar responds immediately by creating the relevant shape or event on the "3D scene".

The "Editor" command area is divided into two parts:

- the area where the instructions to be executed by the avatar are written in a symbolic way (upper part) and
- the area where responding messages are automatically written by MaLT2 environment in accordance to the realized actions (lower part). These messages refer either to an error in the structure of commands or to the correct definition of a new procedure, and they appear as feedback and troubleshooting guide for the user.

By using the "Editor" component, you may execute commands as follows: place the cursor on the line where the commands you want to execute are located. Press the key 'insert', usually found in the keyboard with the initials letters 'ins'. Each time you press the key 'insert' Logo executes the words of the line on which the cursor is placed from left to right. When a command is not recognized, either due to the fact that it has not been defined or because it does not belong to the basic commands, the message 'I don't know how to' appears. You can also do the same action by clicking the play button  at "Editor's" toolbar.

Note: When you press the usual key 'enter', the cursor simply changes line without executing any commands.

1.2.3 The Variation tools

With the 'Variation tool' and the '2D Variation tool' components you may provoke dynamic constant change to the shapes created by the avatar when a parametric command that you have defined has been given. (More information about the definition of new commands in [section 4.2](#)). The variation tools cannot be activated for stable commands (without variation as an input) and for basic commands (even with an input). To activate and display this tool, you should click on the trace of the entity formed by a command.

The 'Variation tool' component changes in a constant way the value(s) of input variables that you have set for a command. At the same time, the shape also changes dynamically. This occurs when you drag the cursor over the variation slider. With the '2D Variation tool' you may see what happens in the shape, as you co-vary two variables on imaginary vertical axes, by freely dragging the cursor on the interface of the component. (More information about the variation tools in [section 5.](#))

2 Features of the 3D scene

This section describes the main functions of the 3D scene of MaLT2.

2.1 Camera controls

As the environment of MaLT2's scene is represented by three dimensions, a perspective camera is available and you can look at the scene from different points of view in the 3D space. The default position of the camera is in alignment with the scene's center, resulting in a 2D view of the scene's representations. However, if you rotate the camera to any direction, you will have a 3D view of the scene. As the 3D scene is in a sphere shape, the camera moves on the circumference of that invisible sphere.

You can control the camera by clicking anywhere in the scene's area and then by using one of the following ways of control:

1. Use the following keys of your keyboard:

W or ↑: Camera turns up

S or ↓: Camera turns down

A or ←: Camera turns left

D or →: Camera turns right

E: Camera zooms in

Q: Camera zooms out



2. Use the mouse/mousepad:

Hold the left click pressed and move the mouse to rotate the camera along with the mouse movement.

Hold the mouse wheel pressed and move the mouse forward: Camera Zooms in.

Hold the mouse wheel pressed and move the mouse backwards: Camera Zooms out.


3. Use the camera control tools at the left bottom corner of the scene.


Click on the arrows  to rotate the camera on the circumference of the sphere.
Click on the buttons  to zoom in/zoom out the camera.


Note: You can reset the camera to its default position by clicking the button .


2.2 Scene's toolbar


At the top of the 3D scene there is a toolbar with a group of tools related to the scene or the avatar. The functions of each tool are described below:

 Eraser: It erases all the graphics drawn on the scene, it returns the avatar to its initial position (0,0,0) and the camera to its initial rotation.

 Target: It locates the avatar on the screen and moves the camera to a position where the avatar is visible. (It is used when the avatar has moved out of the camera's field of view).

 Axes: This button shows or hides three help axes (x,y,z) on the avatar.

 Help: At help menu you can find a link to this manual of MaLT2, a link to a list of all the available Logo commands you can use in MaLT2 environment and also some examples of models made in MaLT2 (ex. a cube, a pyramid).

 This button gives you the option to change the object of the entity (hummingbird, airplane, etc.), the background color of the scene, and the color of the trace (pen) that the entity leaves.

3 Basic control – avatar guidance

This section describes the way to control and guide the avatar through symbolic instructions that are created in the editor area.

3.1 What is a command?

As mentioned before, the avatar is controlled with the application of the commands written in the "Editor". The command is the symbolic expression of an instruction which leads to a specific result when it is executed. Each command has a unique name and it is composed in a predefined way. A command may be simple, i.e., it can be composed only by using its name and execute a specific action, or it may include input and output parameters. These parameters consist of numbers, words or other data. An input command consists of such data that are indispensable for its execution. An output command is the outcome after its execution. A command may require **none, one or many inputs** whereas it can have **none or one output**.

In particular, the commands in MaLT2 are composed as follows:

CommandName(space)input1(space)input2 etc.

e.g., forward 100

Some basic MaLT2 entity control commands will be presented in the next chapter, as well as how they are composed and executed.

3.2 Movement of the avatar on the surface

In order to move the avatar on "3D scene" a number of movement commands can be used. These commands define the way and the degree of avatar's movement. For the avatar to move **forward** by a specific number of steps the command *forward* should be used. The command *forward* has an input which should be a number and it defines the steps that the avatar will proceed. This number is written straight after the command. The result of the command is that the avatar moves towards the direction of its head in a distance of as many steps as the value of the number in the command input. As the avatar moves, it leaves the relevant trace behind. For example, the command ***forward 50*** asks the avatar to move 50


steps forward. For any command written in the “Editor” that you want to be executed, either the button ins (INSERT) or the button  located on the command editor toolbar should be clicked, while the editor writing indicator (cursor) is on the line of the command.



Figure 3: Avatar moves forward 50 steps

The *back* command which asks the avatar to move a number of steps towards the opposite direction from the direction of its head, operates in exactly the same way.

Example:

back 50

For the avatar’s rotation and direction there are six different commands. These commands take as input a number which defines the degrees according to which you wish to turn the avatar’s head. These six commands are the following:


Right and Left: The avatar turns its head to the right or to the left as much degrees as the number given as input. For example, the command ***right 90*** asks the avatar to turn its head 90 degrees to the right. In the same way, the command ***left 30*** asks the avatar to turn its head 30 degrees to the left.

Up and Down: The avatar turns its head upwards (look up) or downwards (look down) as much degrees as the number given as input. For example, the command ***up 90*** asks the avatar to turn its head 90 degrees upward. In the same way, the command ***down 30*** asks the avatar to turn its head 30 degrees downwards.

Roll_Right and Roll_Left: The avatar rotates around itself to the right (clockwise) or to the left (anticlockwise) as much degrees as the number given as input. For example, the command ***roll_right 90*** asks the avatar to rotate around itself 90 degrees clockwise. In the same way the command ***roll_left 30*** asks the avatar to rotate around itself 30 degrees anticlockwise.

Important tips

1st Tip: Pay attention when composing a command! Between the name of the command and its input there should be a **blank** space. For example, if the command ‘forward 50’ is written without a blank in between (‘forward50’) the message on the editor will be «I don’t know how to forward50» because it does not recognize any command with such a name.

2nd Tip: Each command may be executed several times as long as the cursor is placed on the line where the command is written and the button “ins” or the button  is clicked. For example, you can execute the command *forward 10*, then the command *back 30*, and then again, the command *forward 10*.

3rd Tip: Apart from the individual execution of commands, as mentioned above, there is the possibility of executing numerous commands simultaneously. The mode of executing

commands is **per line**. In particular, there are two modes of simultaneous execution of commands.

1st mode: Serial execution on different lines (from top to bottom).

Let's say you want to execute the following two lines of command together:


```
forward 50
```

```
forward 30
```

First you select them:

```
forward 50
```


```
forward 30
```

and then you click on the key "ins" or the button  for MaLT2 to run them in the same order from top to bottom.

2nd mode: Serial execution in one line (from left to right).

Another way to run the above two commands is to write them on the same line as follows:


```
forward 50 forward 30
```

They are then executed in the same way as one command on the line, from left to right, by placing the cursor on the line and pressing the key ins or the button .

(**Pay attention to the spaces** between the commands and their value inputs!).

4th Tip: The commands may take as an input integer, decimal, positive and negative numbers.

5th Tip: Note that the command *forward -20* has the same outcome as the command *back 20*!

6th Tip: You can undo the last command you executed by clicking the button  of the "Editor's" toolbar. (This means that the button cancels the last execution of commands, but it does not change the commands written in the "Editor" window.)

3.3 Calculations using MaLT2

The commands in MaLT2 allow to perform calculations within the commands.

For example, the command:

```
forward 50+50
```

moves the avatar 100 steps forward towards the direction of its head. The program perceives that the command *forward* has as an input the result of the addition $50+50$. The same happens with more complex calculations such as:

```
forward (50-20)*3/2
```

The parentheses rules in mathematics apply in complex calculations. MaLT2 perceives the calculations as separate commands as well. In other words, instead of using numerical symbols, there are commands that execute the calculation.

Eg.: $30+20$ is also perceived as **sum** 30 20
 $30-10$ is also perceived as **difference** 30 10
 30×10 is also perceived as **product** 30 10
 $30/10$ is also perceived as **divide** 30 10

The above mode -to firstly denote the name of the result and then the numbers which are involved in the calculation- is very useful when you wish to define calculations as the following:

Eg. 2^3 as **power** 2 3
 $\sqrt{5}$ as **root** 5

For example, you may execute the command: *forward root 400*.

For more mathematical commands see Table 2 in [Appendix A](#).

Note: Calculation commands cannot be executed on their own in the "Editor". For example, if the *root 36* is run nothing will happen. This happens because the calculation commands have one output, but the result of the calculation must be applied somewhere. If you want to see the result of a mathematical command without using it in another command, you have to use the command **print**. The print command, displays at the message area anything that is given as input to it. So, you write a mathematical command after the print command to print its result to the message area.

E.g., if you execute the command *print root 100* the message "10" will be displayed in the message area, where 10 is the result of *root 100*.

3.4 The avatar's trace

As mentioned above, when the avatar moves on the '3D scene', it leaves a trace. This trace can be controlled by a number of commands. For example, you can define whether the avatar leaves a trace as it moves or not. This is realized with the commands *pendown* and *penup*, respectively. After the command *penup* is executed, the avatar leaves no trace as it moves, whereas in the case of the command *pendown*, the avatar leaves a trace behind as it moves. Furthermore, there is the command *clearscreen (cs/cg)*, which deletes anything that the avatar has designed so far on the '3D scene' and restores the avatar to its initial position (0,0,0) with its head turned upwards, as it was initially. It also resets the camera to its default position.

Finally, there are several commands regarding the change of color and density of the avatar's trace, included in table 1 of [Appendix A](#).

4 Structural language features in MaLT2

4.1 Primitives

MaLT2 includes a number of commands and functions, as the ones described above (*forward*, *clearscreen*, *right* etc.). These commands are called **primitives** and they possess the following features:

- They run either a command or a function.
- They may take value inputs or not.
- They may have an output value or not.

Some basic typical primitives are presented below:

Procedure	Number of inputs	Type of input data	Outcome-event
clearscreen	0	-	Clears the 'Canvas' and restores the avatar in its initial position
right x	1	number	Turns the avatar's head x degrees right
up y	1	number	Turns the avatar's head y degrees upwards
penup	0	-	Raises the avatar's pen

Tip: The commands also have abbreviations. MaLT2 can understand the commands even with their names written abbreviated. E.g., the command *clearscreen* as *cs*, the command *right 30* as *rt 30*, the command *left 30* as *lt 30*, etc.

In the [Appendix](#) of this manual, there is a table with the important procedures as well as examples of their use.

4.2 New commands (Procedures)

An important feature of MaLT2 is that it enables the user to create his/her own commands, which are called 'procedures' and 'sub-procedures' in the IT language. A procedure is a primitive command or a command to which you have given a name of your own choice and you have defined it so that it runs a number of commands. In other words, MaLT2 allows you to create your own additional words-commands besides the existing primitives and use them wherever and however you wish.

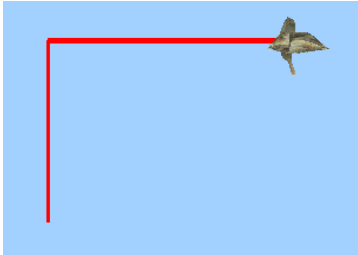
Every time you define a procedure-command it has the same properties with the primitive commands of MaLT2. Thus, you can define a command by using another command you have already defined. In this way, you create a limitless structure of procedures and sub-procedures.

4.3 Procedure construction

Let's say you have edited the following commands:


```
forward 60  
right 90  
forward 80  
left 90
```

If you run these commands in this order, the following shape is created on the 3D scene:



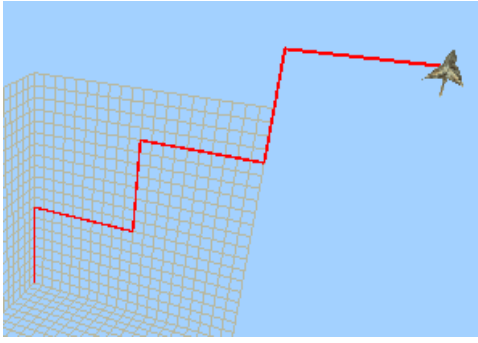
You can define a procedure that will run the above commands in the same order every time that it will be called. In order to do this, you give the command **"TO"** and a name of your choice to the procedure. In a separate line, after the end of the series of commands that the new procedure will require, it is necessary to enter the word-command **"END"**. Don't forget it! In other words, the code will be:

```
TO stair  
forward 60  
right 90  
forward 80  
left 90  
END
```

The word **"to"** is a primitive MaLT2 command which is used for the definition of a new procedure. On the right of the word "to", the word you have chosen to call the new procedure is written (in this particular case *stair*). On the last line the word **"end"** is written and it informs the system that the procedure initiated with the word "to" has ended. To complete the definition of the new procedure you must select all the lines and press the key ins (insert) or the button . Then, on the lower part of the "Editor", there will appear the message «stair defined», which means that the procedure *stair* has been defined. From now on, you can use the word *stair* as a command. For example, if you write on the 'Editor':

```
stair
```

and run the command, the 4 commands will automatically be executed and the above shape will be created on the '3D scene'. Now that the procedure *stair* has been defined, you can run the procedure as many times as you wish. For example, if you run the procedure 3 consecutive times, then the following shape will occur:



Attention: A procedure can be named according to your wish, as long as:

1st: It is only one word. The name *stair up* is not acceptable because it consists of two words separated with a space. The name *stair_up* is acceptable.

2nd: It does not contain mathematical symbols (+ - / * etc.)

3rd: It is not a name that corresponds to a primitive MaLT2 command.

For example:

Acceptable names: *two_squares, 2squares, square2*

Not acceptable names: *two squares, two-squares, forward*

For your own convenience it would be better for the names of the commands to be relevant with what the procedure denotes.

4.3.1 Procedure inputs

The procedures can have inputs and outputs the same way as MaLT2 commands. This is realized by the use of **variables**. The variables that a procedure wants to receive should be defined to the right of the procedure's name when defining it.

By using a variable, the above procedure can be realized:

```
TO stair :height
```

```
forward :height
```

```
right 90
```

```
forward 80
```

```
left 90
```

```
END
```

The height of the stair is now **variable** and can be defined by the user during the execution of the procedure. For example, if you run the command as *stair 100*, a stair is created with a height of 100 steps.

Attention 1: Every time you use a variable in the code, the symbol **:** must precede its name. After **:** do **not** leave a space! When you use a variable in the right way, its color turns to blue.

Attention 2: The names of the variables must also be a consecutive word and do not contain mathematical symbols (+ - / * etc.).

Another example of the use of variables is the following:

```
TO stair :height :width
forward :height
right 90
forward :width
left 90
END
```

In this example both the height and the width of the stair are variable and defined by values that are given as inputs at the execution of the command. This procedure can be run as **stair 40 80** and it creates a stair with a height of 40 steps and a width of 80 steps. It is important that you write the variable values in the order they were defined in the procedure; in the specific example first write the value of the height and then the value of the width.

Advice: A procedure may have as input as many variables as you wish.

4.4 Sub-procedures & Hyper-procedures

MaLT2 allows for other procedures to be called within the procedures. Let's suppose that we have define the aforementioned procedure *stair :height :width*.

You may define a new procedure which will execute the following:

```
TO staircase :height :width
stair 30 40
stair :height :width
END
```

The new procedure *staircase* calls *the* procedure *stair* twice as a standard command. The first time it is called with stable values whereas the second time with values that the procedure *stair* itself takes as an input.

In this particular case, the procedure *stair* is called **sub-procedure** and the *staircase* **hyper-procedure**.

Another example is the following:

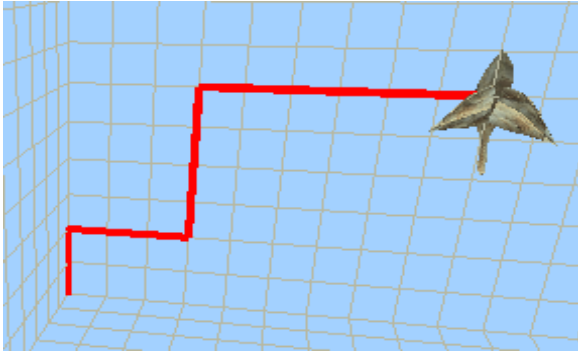
```
TO doubleStairs :height :width
stair :height :width
stair :height*2 :width*2
END
```

The procedure creates initially a stair with the height and the width that you have defined as parameters and then one more stair with double height and double width.

The outcome from the execution of the above procedure as:

```
doubleStairs 20 30
```

is the following:



Advice: In a hyper-procedure, such as the *stair*, you can call many different procedures. For example, if you had also defined a procedure *stair_up*, you could have called apart from this one the procedure *stair* as well within the procedure *staircase*.

5 Dynamic manipulation – Variation tools

In this section, the other two components of MaLT2 are described; the 'Variation Tool' and the '2D Variation Tool'.

5.1 The Variation Tool

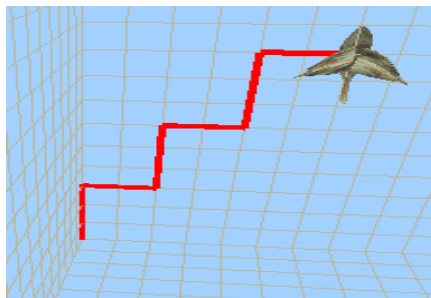
The 'Variation Tool' allows the user to **dynamically** manipulate the variables of a function that has been defined.

For example, let's suppose you have defined the following procedure:

```

TO staircase :height :width
  stair :height :width
  stair :height :width
  stair :height :width
END
    
```

The procedure *staircase* calls 3 times the sub-procedure *stair* and creates three consecutive stairs. It has two variables; *:height* and *:width*, which define the height and width of the stairs. If you run the procedure as *staircase 30 20*, the avatar draws the following shape:



If you move the mouse into the '3D scene' you can left click on any part of the character's (red by default) trace. Clicking on the trace (it turns green) indicated in the above example, the 'Variation tool' (the blow area below the 'Editor') acquires two sliders (Figure 4).

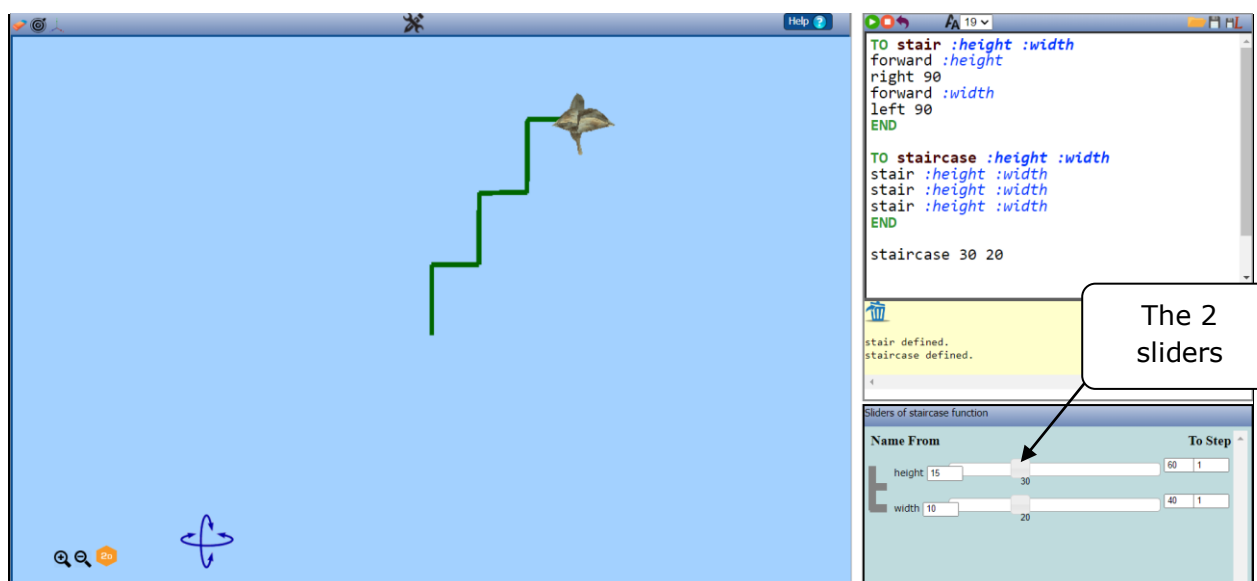


Figure 4: The variation tools corresponding to the height and width variables

In particular, these sliders correspond to the two variables of the *stair* *:height* *:width* procedure and initially they have values according to which the procedure was run, namely, *:height* = 30 and *:width* = 20. These sliders allow the user to **dynamically** change the variable values by moving the respective indexes and automatically observe the changes that take place on the avatar's trace.

On the left and on the right of the variation *height* toolbar there are two fields: "From" and "To" which contain the numbers 15 and 60, respectively. These numbers constitute **the limits** within which the *height* variable values change. You can change these limits and write in the respective fields the numbers of limits that you wish. There is one more field called "Step" which contains number 1. This entails that the variation tool can take values which differ by one unit between the limits you have defined. You may change this as well by applying the number you wish.

The concept of the variation tool can be applied for the formulation of open-ended problems, such as:

- Move the indicator that corresponds to the height variation tool and observe the way the staircase inclination changes.
- Move the indicator that corresponds to the width variation tool and observe the way the staircase inclination changes.
- Try to figure out the relation between the *height* and *width* variables in order for the staircase to maintain its inclination.

5.2 The 2D Variation Tool

The '2D Variation Tool' allows you to represent two of the variables of a specific procedure on **an orthonormal system of co-ordinates**.

Let's suppose that you have defined the *staircase* procedure, as above, and that you have already activated the 'Variation tool'.

TO *staircase* *:height* *:width*

stair *:height* *:width*

stair *:height* *:width*

stair *:height* *:width*

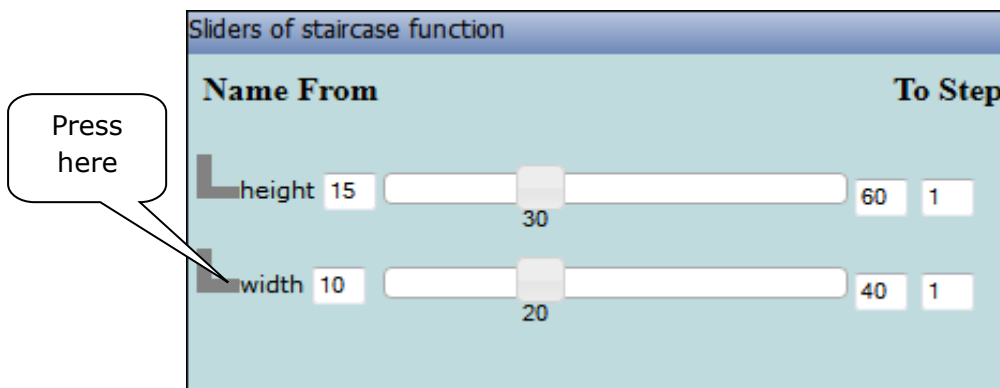
END

On the left side of each slider there is a grey right angle. By clicking on either side of the angle you can set this variable on the corresponding axis of the rectangular axis system, represented by the '2D Variation Tool'.

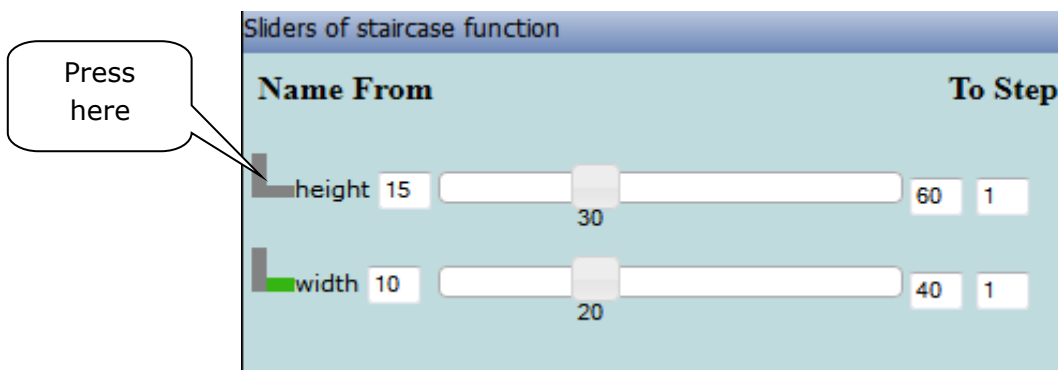
For example, let's suppose that you want to create an axis system where the vertical axis will be the stair height and the horizontal axis will be the stair width.

To do this, the following procedure must be followed:

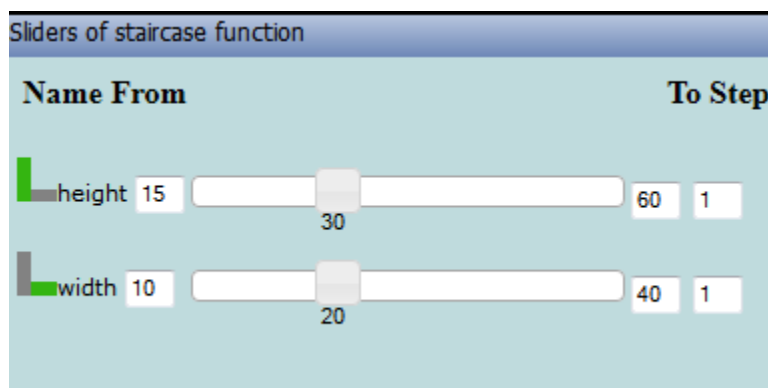
Allocation of the *width* variable to the horizontal axis.




Allocation of the *height* variable to the vertical axis.



The selected sides have changed color and turned to green.



On the '2D Variation Tool' an orthonormal system with perceptible axes has now been created, where the vertical axis corresponds to the *height* variable and the horizontal one to the *width* variable of the *stair* procedure. By pressing the key  of the '2D Variation Tool' you can create checkpoints on the axis system. By holding the key pressed and by clicking on any point of the variation tool surface, a red point is created with specific coordinates. The height and width variable values have automatically taken the coordinate values of this point (Figure 5). Moving the red point on the imaginary axis system both values of the height and width variation tools change automatically. If you choose to drag an already placed point, you click on it and hold it pressed and then its color changes from red to blue.

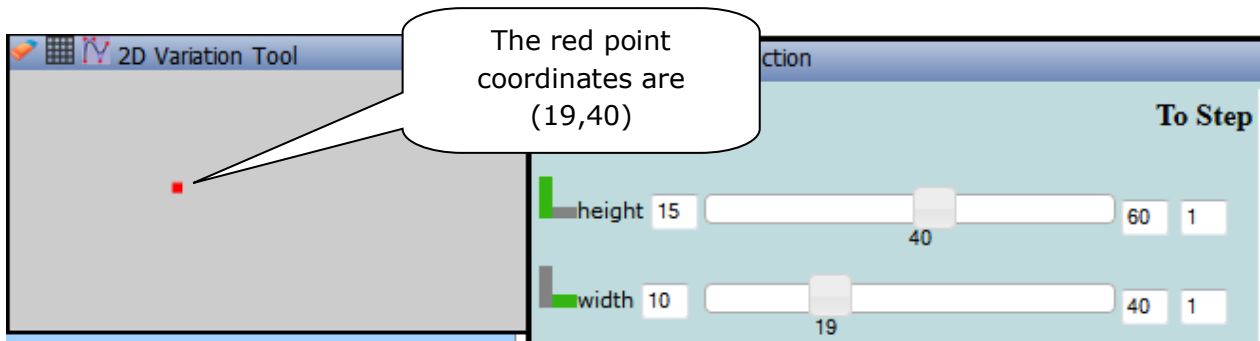





Figure 5:A point on the "2D Variation Tool" in MaLT2

The button  of the '2D Variation Tool' projects a grid on the axis system whereas the button  «clears» the variation tool from all the points that have been created.

Free dragging

If the button  is not pressed, by dragging the mouse on the surface of the '2D Variation Tool' with the left key of the mouse continuously pressed, you can design lines which correspond to the changes occurring on the shape created by the avatar.

5.3 An example

Suppose you want to study the fact that the maintenance of the staircase inclination demands that the ratios of the height and width sizes remain stable. The '2D Variation Tool' helps on the study of the two sizes graphical relation and the inferences concerning the inclination and graphs of similar amounts. Suppose the ratio equals 2, in other words $height/width=2$. Try to place the red point on the '2D Variation Tool' in a position where the $height/width$ ratio equals two. If you place some more red points in different positions where the ratio of the two variables equals two, the following point set-up will occur:

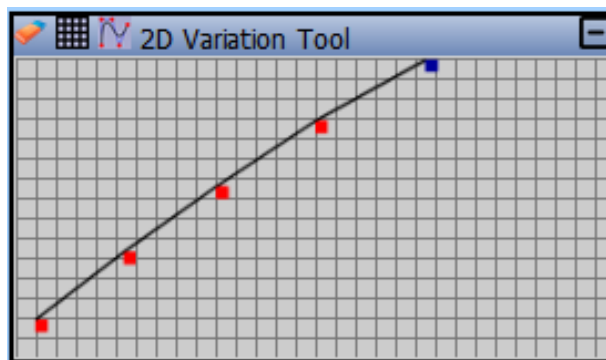


Figure 6: The 2D variation tool

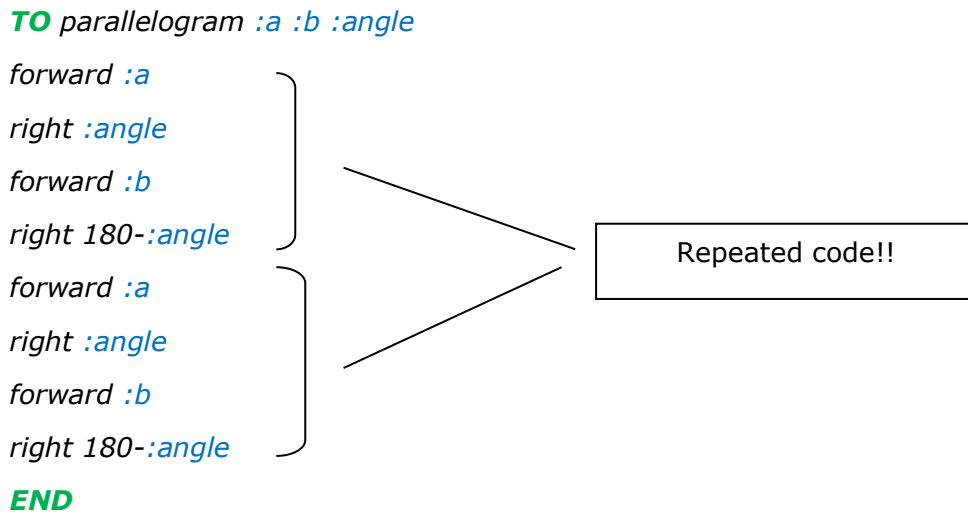
Notice that the points make a straight line with a stable inclination. This line also constitutes the $height/width=2$ graph function.

The rationale of the '2D Variation Tool' can be applied to the formulation of open-ended problems, such as:

- Move the mouse freely on the surface of the '2D Variation Tool' and try to understand from the created trace the way that the variables correlate in order for certain conditions to be met.

6 Repetition structure

MaLT2 allows the use of a repetitive structure in the editor's code. This structure is a primitive which is used for the better flow and organization of the code commands. Let's suppose that you want to define a procedure that results to the formation of a variable shaped parallelogram. One way to do this is the following:



This code defines the *parallelogram* procedure whose commands formulate a parallelogram. With the use of the repetition structure, the repeated code can be written only once. Therefore, the above code becomes:

```

TO parallelogram :a :b :angle
repeat 2 [forward :a
right :angle
forward :b
right 180-:angle
]
END
    
```

The word "**repeat**" is a MaLT2 primitive repetition command. The number that follows it denotes the times the command is repeated (in the specific case 2). The commands inside the brackets **[]** are the ones the avatar will repeat as many times as defined by the number. Generally, the repetition structure is defined as:

```

repeat repetition_times [commands_to_be_repeated]
    
```

The repetition structure is very useful since its application lengthy codes with repetitive commands are avoided. Therefore, it helps in the syntax of a legible and structured code.

Attention!! It is mandatory to close every open bracket.

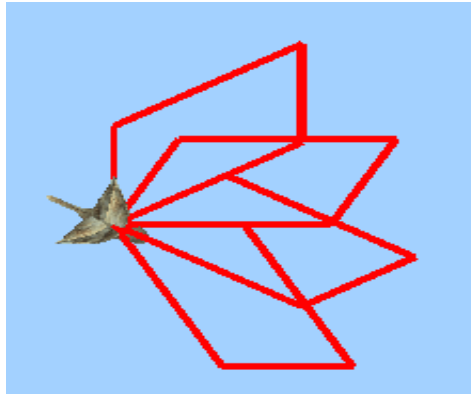
The commands called upon in the brackets could be any MaLT2 commands procedures defined by the user. Moreover, the number of repetitions could be a variable. For example, the following procedure could be defined:

```
TO parallelograms :times :a :b :angle  
repeat :times [parallelogram :a :b :angle right 30]  
END
```

The *parallelograms* procedure repetitively calls the *parallelogram* procedure, as set above, as many times as the variable value *:times* defines. Thus, for example, the execution:

```
parallelograms 4 30 50 60
```

has the following outcome on the '3D scene':



Another example is the creation of a cube, by using four squares. First, we define the procedure *square :a*, which executes 4 times the commands: *forward :a right 90*

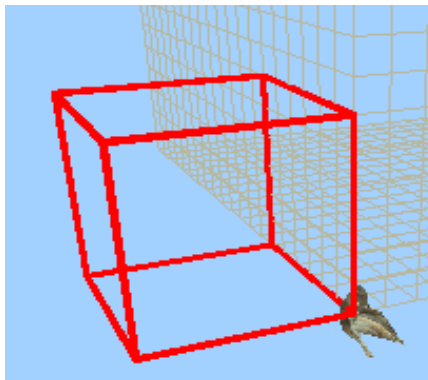
To do that we write at the editor:

```
TO square :a  
repeat 4 [forward :a right 90]  
END
```

The procedure *square* creates a *square* of variable size. Then, we define the procedure *cube :a* like this:

```
TO cube :a  
repeat 4 [square :a forward :a down 90]  
END
```

The procedure *cube :a* creates a cube consisted by 4 squares. This is done by repeating 4 times the procedure *square :a*, then move forward the number of the side and, then turn downwards 90 degrees. If we run the procedure *cube* as *cube 60* the avatar will draw the cube shape showing in the next picture:



7 Recursive procedures

In the *hyper-procedures* and *sub-procedures* section, it was described the way one procedure is called within another procedure. The Logo programming language also allows for a procedure to call itself. This is called "recursion".

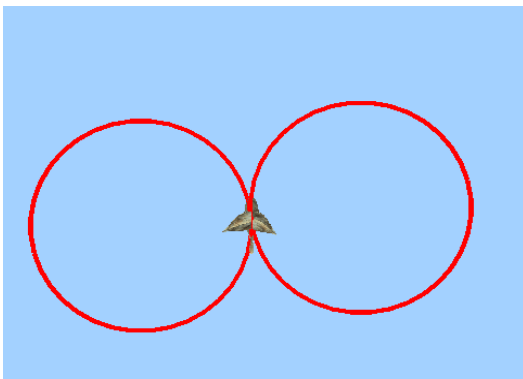
Let's suppose that you have defined the following procedure concerning the formation of a circle with radius r :

```
TO circle :r  
  repeat 36 [forward (2*pi*:r)/36 right 10]  
END
```

and, you have also defined the procedure:

```
TO twocircles :r  
  repeat 2 [circle :r rt 180]  
END
```

which formulates two externally tangent circles, as shown in the following figure:



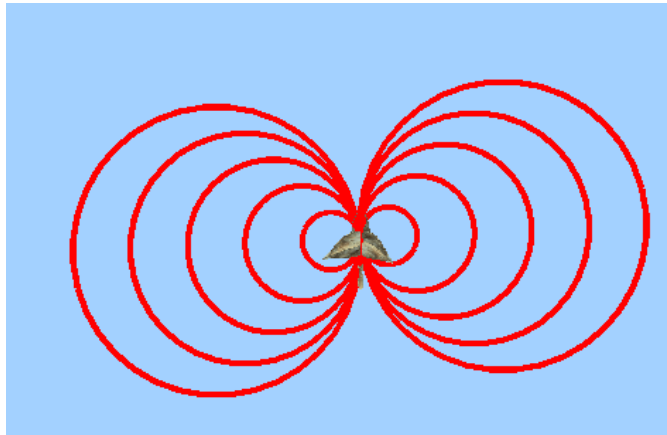
An example of the use of recursion in this procedure, is the following:

```
TO butterfly :n :r  
  if :n < 1 [stop]  
  repeat 2 [twocircles :r rt 180]  
  butterfly :n-1 :r-10  
END
```

Recursion: The procedure calls upon itself!

This *butterfly* procedure creates two tangent circles and it is **calling upon itself** by applying a radius reduced by 10 ($:r-10$). This is done for $:n$ times.

Therefore, if you run the procedure as *butterfly 5 50*, the avatar creates the following shape:



8 Conditional commands

In the *butterfly* example of the previous section, the following command was found within the code of the procedure:

```
if :n < 1 [stop]
```

This is a MaLT2 control command. These commands allow you to check the code execution flow of a procedure based on certain conditions that you define. The “**if**” command checks the **condition** that follows right after (here `:n < 1`). If the condition is true, the procedure runs the commands written in the brackets **[]**.

Let’s define the following procedure:

```
TO stair :height :width  
if :height < 5 [stop]  
forward :height right 90 forward :width left 90  
END
```

The command *if* checks at this point if the condition `:height < 5` is true. If it is true then it will run the *stop* command, otherwise, if it is false, it will ignore the brackets and continue normally with the execution of the following commands. The *stop* command is a MaLT2 primitive command which stops the execution of any procedure run at the same time. Thus, if the height has a value less than 5, the procedure will immediately stop and the commands concerning the formation of the stair will not be executed.

So, for example, if it is executed as:

```
stair 4 10
```

the avatar will not do anything, since the condition is true.

But if it is executed as

```
stair 6 10
```

the avatar will create a stair.

Note: An inequality or an equation of any two elements can be applied as a condition.

Condition examples:

```
:height = 5
```

```
:height > :width
```

```
:height + 3 < 20
```

8.1 Control commands in recursion

The control commands are very useful in the recursive procedures. For example, let’s keep the *butterfly* procedure of the previous section:

```
TO butterfly :n :r  
if :n < 1 [stop]  
repeat 2 [twocircles :r rt 180]  
butterfly :n-1 :r-10  
END
```

If there wasn't the *if* command, the *butterfly* procedure would call upon itself to infinity. By applying control commands you define when the execution of a recursive procedure stops. Thus, if you run the *butterfly* procedure as *butterfly 5 50*, MaLT2 runs the *butterfly* procedure with the values *:n=5* and *:r=50*. The *if* control condition checks whether *:n* is less than 1. When it gets less than 1, the execution of the procedure will stop. For the time being, this is not true and therefore the execution continues formulating two tangent circles with a radius of 50. Next, it calls upon itself with the *:n* value reduced by 1 and the *:r* value reduced by 10. In this case, *:n=4* and *:r=40*.

The execution continues in the same way until it calls upon itself for *:n=0*. Then the control command will be true and the procedure will stop after it has been run for 5 times (as many as the *n* initial value).

The following table shows in detail the *:n* and *:r* values during all the recursion running stages:


Execution flow	Value :n	Value :r	Control condition :n < 1	Recursion command values (butterfly :n-1 :r-10)
1 st	5	50	FALSE	<i>butterfly 4 40</i>
2 nd	4	40	FALSE	<i>butterfly 3 30</i>
3 rd	3	30	FALSE	<i>butterfly 2 20</i>
4 th	2	20	FALSE	<i>butterfly 1 10</i>
5 th	1	10	FALSE	<i>butterfly 0 0</i>
6 th	0	0	TRUE	-

9 Other MaLT2 features


9.1 Save files

In MaLT2 you can save your work in a file at your computer. Then, you may open this file again in MaLT2 and continue your work from the point you have saved it.


There are two different ways to save a file locally:

1. You can save **only the Logo Code** of the 'Editor' by clicking the button  of "Editor's" toolbar. In that case, you will save only the text that is written in the 'Editor' area. The file will be downloaded at the folder where the downloads of your browser are being saved. (Usually it is the folder 'Downloads').

When you open this file again in MaLT2, the code you have saved will be loaded at the "Editor".

2. You can save the **whole state** of the MaLT2 by clicking the button  of 'Editor's' toolbar. Actually, when saving the whole state, the following attributes will be saved:


- a) The Editor's contents (Logo code, text etc.).
- b) The current trace on the scene.
- c) The current camera's position.
- d) The sliders of the variation tool (if they are activated) and their current values.
- e) The notes of the notepad.
- f) The current avatar of the scene (bird/airplane etc.).

The save of the state described above is done with the button  (save all). The file will be downloaded at the folder where the downloads of your browser are being saved. (Usually it is the folder 'Downloads').

When you open this file again in MaLT2, all the above elements will load the state they had at the time you saved the file.

Attention: When you click either of the save buttons, a pop-up window appears and asks you to give a name to the file you want to save. Type the filename you want and then click the OK button.


9.2 Open files


You can open saved files of both types as described above (just Logo code or whole state) by clicking the button  of 'Editor's' toolbar. When you open a file all the saved attributes are being restored to the state they were saved.

9.3 Other features

Some other features of MaLT2 environment are described below.


Notepad

MaLT2 has a notepad where you can keep free notes. To make the notepad visible click the button . As mentioned before, your notes can be saved together with the whole state of the MaLT2 by clicking the save all button.

At the same time, there is another window that opens when you press the button  to the left of the scene, where you can highlight various tips (or rules) that refer to another user who will use what you have built in MaLT2.

Windows manipulation

You are able to move the windows of the "Editor", the variation tools and the notes, wherever you want in MaLT2 environment. To move a window, press the left click on the blue bar on its top and while holding the left click pressed, move the window at any position you want and release the left click (drag and drop). You may also change the dimensions of those windows according to your preferences.

By clicking the button  which is located at the left toolbar of the 3D scene, you can restore the windows to their initial positions and dimensions.

Language settings

MaLT2 is available in both Greek and English language. You can change the language by clicking the language button at the left top corner of the MaLT2 environment.

Appendix A – Tables of Commands

“MaLT2 Commands”

Table 1: Avatar’s control commands

Command	Description	Example
Avatar’s movement		
forward/fd <i>number</i>	Avatar moves forward as many steps as the <i>number</i> value.	fd 50
back/bk <i>number</i>	Avatar moves backward as many steps as the <i>number</i> value.	bk 70
Avatar’s orientation		
right/rt <i>number</i>	Avatar turns its head to the right by as many degrees as the <i>number</i> value.	right 90
left/lt <i>number</i>	Avatar turns its head to the left by as many degrees as the <i>number</i> value.	lt 120
up <i>number</i>	Avatar turns its head upwards (looks up) by as many degrees as the <i>number</i> value.	up 50
down/dn <i>number</i>	Avatar turns its head downwards (looks down) by as many degrees as the <i>number</i> value.	down 60
roll_right/ rr <i>number</i>	Avatar rotates around itself clockwise by as many degrees as the <i>number</i> value.	rr 40
roll_left/rl <i>number</i>	Avatar rotates around itself anticlockwise by as many degrees as the <i>number</i> value.	rl 30
Avatar’s position		
setx <i>number</i>	Places the avatar at the position where x coordinate equals to the	setx 100

	<i>number.</i>	
sety <i>number</i>	Places the avatar at the position where y coordinate equals to the <i>number</i> .	sety -50
setz <i>number</i>	Places the avatar at the position where z coordinate equals to the <i>number</i> .	setz 90
setxy <i>n1 n2</i>	Places the avatar at the position where x coordinate equals to the <i>n1</i> and y coordinate equals to <i>n2</i> .	setxy 50 100
setxz <i>n1 n2</i>	Places the avatar at the position where x coordinate equals to the <i>n1</i> and z coordinate equals to <i>n2</i> .	setxz 50 -90
setyz <i>n1 n2</i>	Places the avatar at the position where y coordinate equals to the <i>n1</i> and z coordinate equals to <i>n2</i> .	setyz 50 -90
setpos [<i>n1 n2 n3</i>]	Places the avatar at the position with the coordinates <i>n1 n2 n3</i> .	setpos [0 30 70]
xcor	Returns the value of the x coordinate of avatar's current position.	
ycor	Returns the value of the y coordinate of avatar's current position.	
zcor	Returns the value of the z coordinate of avatar's current position.	
pos	Returns the avatar's current position in an array of three numbers [x y z].	
distanceto [x y z]	Calculates and returns the distance between the avatar's position and the point given as an array input of [x y z].	distanceto [100 20 30]
Avatar's Trace		
penup/pu	The avatar doesn't leave a trace while moving in the scene.	

pendown/pd	The avatar leaves a trace while moving in the scene.	
setpenseize <i>number</i>	Sets the width of the trace to the value of <i>number</i> . (Default is 1)	setpenseize 5
setpencolor [<i>r b g</i>]	Sets the color of the trace to the color code of the r b g array (red blue green).	setpencolor [0 0 0] (Black)
home	Avatar returns to initial position (0, 0, 0) while leaving a trace.	
cleartrace/ct	Clears the 3D scene and keeps the avatar and the camera in their current position.	
clearscreen/cs/cg	Clears the 3D scene and also resets the avatar to its initial position (0, 0, 0) and the camera to its default position.	
showturtle/st	Shows the avatar on the scene.	
hideturtle/ht	Hides the avatar from the scene.	
Other Commands		
cleartext/ct	Clears messages from the message area.	
print/pr <i>input</i>	Prints the output of the <i>input</i> at the message area. The <i>input</i> may be a command, a mathematical expression or a variable.	print 1+1 print xpos print :height
stop	Stops the execution of the code in a repetition or a recursion. It is necessary in the procedures with recursion!	<u>Example with recursion</u> TO wing :a :n :k if :k < 1 [stop] polygon :a :n wing 2*:a/3 :n :k-1 END

Basic color codes RGB for the avatar's change of color:

Red 255 0 0

Green 0 255 0

Blue 0 0 255

White 255 255 255



You can find more color codes at MaLT2's color picker.

Table 2: Programming structures

Command	Description	Example
Conditional Structures		
if <i>condition</i> [commands]	If the <i>condition</i> is true, the group of commands inside the brackets [] is executed.	if :x > 10 [forward 100 right 90]
ifelse <i>condition</i> [commands1] [commands2]	If the <i>condition</i> is true, the group of commands1 of the first brackets is executed. Else if the condition is false, the group of commands2 of the second brackets is executed.	ifelse :x > 10 [forward 100 right 90] [left 90 forward 100]
if and <i>condition</i> [commands]	If both parts of the <i>condition</i> are true, the group of commands inside the brackets [] is executed.	if and :x>3 :y>5 [fd 100]
Iterative structures		
repeat <i>n</i> [commands]	The group of commands inside the brackets [] is repeated n times .	repeat 4 [forward 100 rt 90]
while <i>condition</i> [commands]	While the condition is true the group of commands inside the brackets [] is repeated.	make "x 1 while :x<5 [fd 100 rt 90 make "x :x+1]
until <i>condition</i> [commands]	Until the condition becomes true , the group of commands inside the brackets [] is repeated.	make "x 0 until :x = 4 [fd 100 rt 90 make "x :x+1]
repcount	Returns the current repetition number. It is used only in "repeat n" structure.	repeat 4 [fd 40 print repcount] It will print 1, 2, 3, 4 in sequence.

Operators		
or <i>Expr1 Expr2</i>	Returns <i>true</i> if at least one of the two expressions is true.	if or 2>3 4<5 [print 'true'] (it is true)
and <i>Expr1 Expr2</i>	Returns <i>true</i> if both expressions are true.	if and 2>3 4<5 [print 'true'] (it is false)
not <i>Expr1</i>	Returns <i>true</i> if Expr1 is not true.	if not 2>3 [print 'true'] (it is true)
equal? <i>Value1 Value2</i>	Returns <i>true</i> if value1 is equal to value2.	if equal? :a :b [print 'equal']
notequal? <i>Value1 Value2</i>	Returns <i>true</i> if value1 is not equal to value2.	if notequal? :a :b [print 'not equal']
greater? <i>Value1 Value2</i>	Returns <i>true</i> if value1 is greater than value2.	if greater? :a :b [print 'a bigger']
less? <i>Value1 Value2</i>	Returns <i>true</i> if value1 is less than value2.	if less? :a :b [print 'a smaller']
greaterequal? <i>Value1 Value2</i>	Returns <i>true</i> if value1 is greater or equal to value2.	
lessequal? <i>Value1 Value2</i>	Returns <i>true</i> if value1 is less or equal to value2.	
make " <i>variable number</i> "	Defines the <i>variable</i> and assigns to the variable the value of the <i>number</i> . Then it can be used as :variable	make "height 30 (:height will have the value 30)
rand/random <i>a b</i>	Returns a random number between a and b-1.	rand 0 4 (returns randomly a number among 0, 1, 2, 3)
output value	Stops the procedure and returns the <i>value</i> . It is used inside procedures.	TO add :a :b return :a + :b END

Table 3: Mathematical Commands

Command	Description	Example	Result
sum/add <i>a b</i>	Returns the sum of the two numbers set in its input, i.e., it	sum 3 5	8

	performs $a+b$.		
difference/sub $a b$	Returns the difference of the two numbers set in its input, i.e., it performs $a-b$.	difference 8 3	5
product/mul $a b$	Returns the product of the two numbers set in its input, i.e., it performs $a*b$.	product 2 4	8
divide/div $a b$	Returns the division of the two numbers set in its input, i.e., it performs a/b .	divide 6 3	2
remainder/modulo/mod $a b$	Returns the remainder of the division of the two numbers set in its input.	remainder 11 2	1
sqrt $number$	Gives the square root of the number set in its input.	sqrt 36	6
power/pow $x n$	Raises the x number to the n power and returns the result. Thus, it is x^n .	power 2 4	16
cos $degrees$	It returns the cosine of the angle set as an input.	cos 60	0.5
sin $degrees$	It returns the sine of the angle set as an input.	sin 60	0.866
tan $degrees$	It returns the tangent of the angle set as an input.	tan 180	0
arccos $number$	It returns the angle that it is calculated by the inverse cosine based on the argument set as an input.	arccos 0.5	60
arcsin $number$	It returns the angle that it is calculated by the inverse sine based on the argument set as an input.	arcsin 0.5	30
arctan $number$	It returns the angle that it is calculated by the inverse tangent based on the argument set as an input.	arctan 1	45
radcos $rads$	It returns the cosine of the angle given in radius (rads).	radcos 1	0.5403023058681 398

radsin <i>rads</i>	It returns the sine of the angle given in radius (rads).	radsin 1	0.8414709848078
exp <i>number</i>	It returns the exponential function with a base of e and a power of the number set in its input (e^{number}).	exp 1	2.718
ln <i>number</i>	It returns the ln value of the number set as an input.	ln 1	0
log10 <i>number</i>	It returns the log10 set as an input.	log10 10	1
integer/int <i>number</i>	It returns the integer part of the number set as an input.	integer 2.8	2
round <i>number</i>	It returns the rounding of the number set in its input.	round 2.3 round 3.8	2 4
minus <i>number</i>	It returns the minus of the number set as an input.	minus 10	-10
abs <i>number</i>	It returns the absolute value of the number set as an input.	abs -3	3
pi	It returns the pi (3,14) number.	pi	3.14