



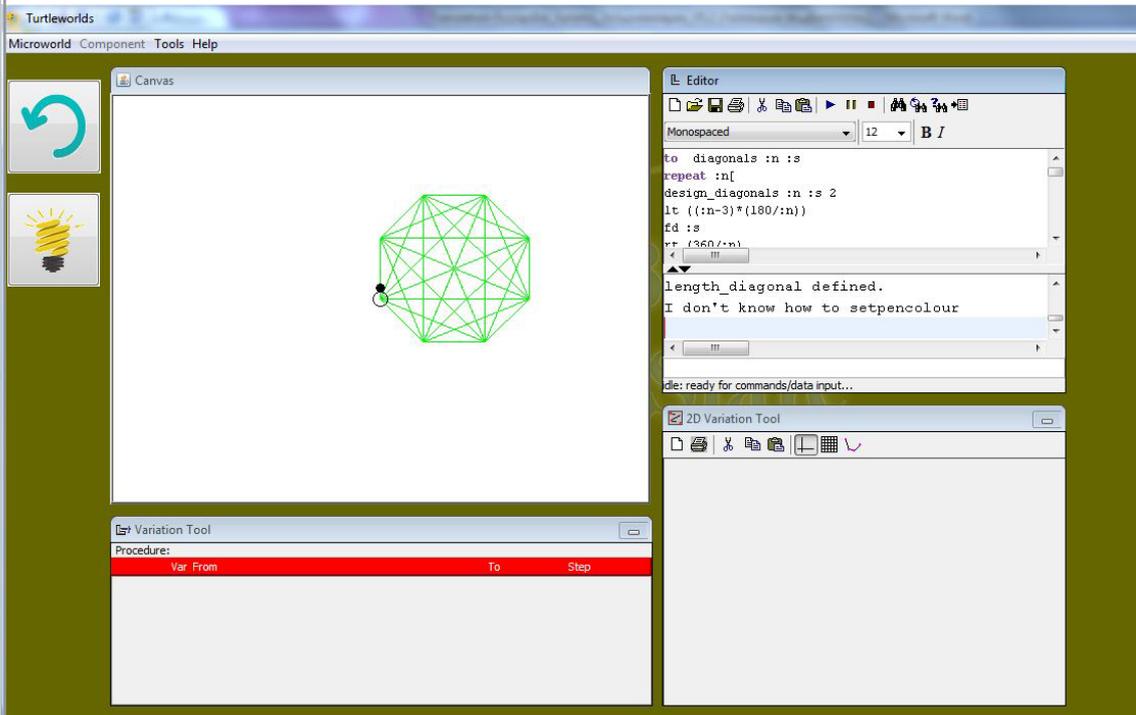
Educational Technology Lab

National and Kapodistrian University of Athens

School of Philosophy

Faculty of Philosophy, Pedagogy and Philosophy
(P.P.P.), Department of Pedagogy

Director: Prof. C. Kynigos



“Turtleworlds” Manual

Ver.: 5.5

TABLE OF CONTENTS

1. Turtleworlds	4
1.1 <i>What is Turtleworlds</i>	4
1.2 <i>Description of Turtleworlds</i>	4
1.2.1 <i>The turtle and the Canvas.....</i>	4
1.2.2 <i>The Editor</i>	4
1.2.3 <i>The Variation tools</i>	5
2 Basic control – turtle guidance.....	6
2.1 <i>What is a command?</i>	6
2.2 <i>Movement of the turtle on the surface</i>	6
2.3 <i>Calculations using Turtleworlds</i>	7
2.4 <i>The turtle’s trace</i>	8
3 Structural language features in Turtleworlds.....	9
3.1 <i>Primitives.....</i>	9
3.2 <i>Procedures – Sub-procedures.....</i>	9
3.3 <i>Procedure construction</i>	10
3.3.1 <i>Procedure inputs</i>	11
3.4 <i>Sub-procedures - Hyperprocedures</i>	12
4 Dynamic manipulation –Variation tools.....	13
4.1 <i>The Variation Tool.....</i>	13
4.2 <i>The 2D Variation Tool.....</i>	14
4.3 <i>An example.....</i>	16
5 Repetition structure	18
6 Recursive procedures	20
7 Control commands	21
7.1 <i>Control commands in recursion.....</i>	21
Appendix A – Tables of Commands.....	23
<i>Table 1: Turtle control commands</i>	23
<i>Table 2: Mathematical function commands in Turtleworlds</i>	24
Appendix B – Additional examples for the use of Turtleworlds.....	27
<i>Example 1: Waves with staircases.....</i>	27
<i>Example 2: Diagonal formation of a canonical polygon with recursion</i>	28
<i>Example 3: Designing the “Rosette” shape by applying recursion</i>	29

1. Turtleworlds

1.1 What is Turtleworlds

Turtleworlds is a tool of symbolic expression in mathematical activity by means of programming for the creation and tinkering of dynamic graphical models.

1.2 Description of Turtleworlds

Turtleworlds consists of four distinct and yet connected work areas. These areas are called components. Each component is defined to run specific activities or functions.

1.2.1 The turtle and the Canvas

On the left side of Turtleworlds appears the component 'Canvas', which also includes the turtle. The canvas is a surface on which the turtle leaves its trace as it moves (unless you prefer it does not leave a trace). The turtle, apart from being a cute pet in Turtleworlds, constitutes a strictly defined mathematical entity. It is defined by its state, in other words by a) its position and b) its orientation. Its position is placed in the centre of the circular object and its orientation is defined by the position of its head. The turtle has a specific trace colour, which is the colour of its head (black by default), as well as specific size, which is adjusted by the settings of its component. You may change all these settings when you get familiar with the construction of models.

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 blog or word processor and b) commands and programs that enable the turtle 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), and 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 turtle and define the changing values of its state attributes; position and orientation. Each time a command is executed, the turtle responds by creating the relevant shape or event on the "Canvas".

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

- the area where the instructions to be executed by the turtle are written in a symbolic way (upper part) and
- the area where responding messages are automatically written by Turtleworlds 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 function as a 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 'ins'. When you press the usual key 'enter', the cursor simply changes line without executing any commands. 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 there is a message projected, saying 'I don't know how to'.

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 turtle when it has been given a parametric command that you have defined yourself. For stable commands without input variation and for basic commands even with an input, the variation tools cannot be activated. 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 4).

The four components can be seen in the following figure (Figure 1):

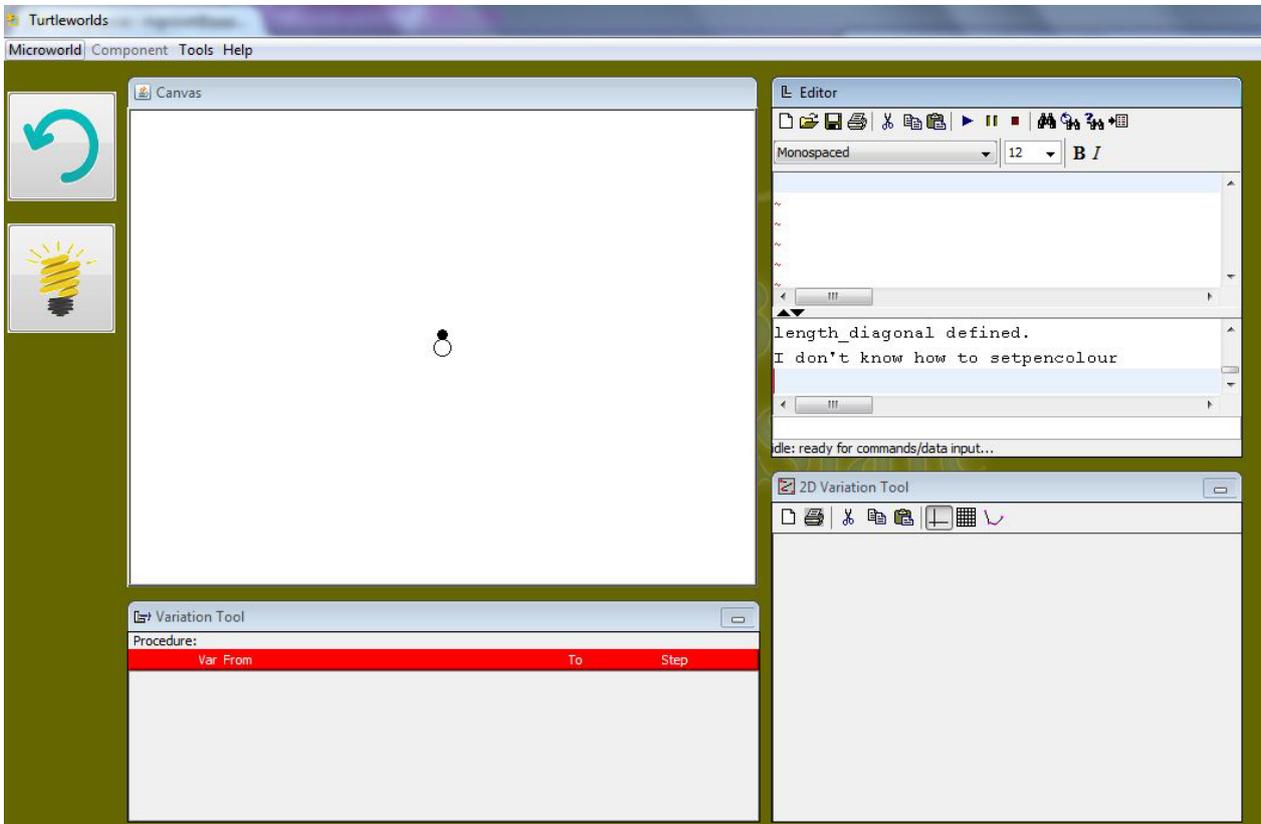


Figure1: Turtleworlds interface

2 Basic control – turtle guidance

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

2.1 What is a command?

As mentioned before the turtle 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 executed. Each command has a unique name and 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 Turtleworlds are composed as follows:

CommandName(space)input1(space)input2 etc.

Some basic Turtleworlds commands regarding turtle control will be presented next together with the way they are composed and executed.

2.2 Movement of the turtle on the surface

In order to move the turtle on “Canvas” a number of movement commands can be used which define the way and the degree of its movement. For the turtle 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 defines the steps the turtle will proceed. This number is written straight after the command. The result of the command is for the turtle to move 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 turtle moves it leaves the relevant trace behind. For example the command:

Forward(space)50

asks the turtle to move 50 steps forward. For any command written in the “Editor” 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 command. The result of executing the specific command is for the turtle to move 50 steps forward towards the direction of its head.

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

Example:

Back (space)50

For the turtle’s **rotation** there are the *Right* and *Left* commands. These commands take as input a number which defines the degrees according to which you wish to turn the turtle’s head. For example the command *Right 90* asks the turtle to turn its head 90 degrees to the right. In the same way the command *Left 30* asks the turtle to turn its head 30 degrees to the left.

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 what to do with 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 may 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.

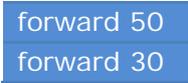
1st mode: Serial execution on different lines (from top to bottom).

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

forward 50

forward 30

First you select them:



and then you click on the key ins or the button  for Turtleworlds 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 was executed (Placing the cursor on the line and pressing the key ins or the button ). Turtleworlds will execute all the commands in the line from left to right (**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.

Note: The command *forward -20* has the same outcome as the command *back 20*!

2.3 Calculations using Turtleworlds

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

For example the command:

Forward 50+50

moves the turtle 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. Turtleworlds 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 ***quotient*** $30\ 10$

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 20*.

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 there will appear the message «You don't say what to do with 6». This means that the command *root* was run with the result of 6 but the program does not recognize what to do with this number. This happens because the calculation commands have one output; the result of the calculation that must apply somewhere. Generally, the commands that have an output cannot be used individually but only as input to other commands.

2.4 The turtle's trace

As mentioned above, when the turtle moves on the 'Canvas', it leaves a trace. This trace can be controlled by a number of commands. For example, you can define whether the turtle leaves a trace as it moves or not. This is realized with the commands *pendown* and *penup*, respectively. The command *penup* turns the turtle's head white, which means that if the turtle moves it will leave no trace, whereas in the case of the command *pendown* the turtle's head will turn black and the turtle will leave a trace behind. Furthermore, there is the command *cleargraphics (cg)*, which deletes anything the turtle has designed so far on the 'Canvas' and restores the turtle to its initial position with its head turned upwards, as it was initially.

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

3 Structural language features in Turtleworlds

3.1 Primitives

Turtleworlds includes a number of commands and functions, as the ones described above (*forward*, *cleargraphics*, *right* etc.). These commands are called **primitives** and 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
Cleargraphics	0	-	Clears the 'Canvas' and restores the turtle in its initial position
Right <i>a</i>	1	number	Turns the turtle's head <i>a</i> degrees right
Left <i>a</i>	1	number	Turns the turtle's head <i>a</i> degrees left
Penup	0	-	Raises the turtle's pen
Pendown	0	-	Lowers the turtle's pen

1st Tip: The commands also have abbreviations. Turtleworlds can understand the commands even with their names written abbreviated. Eg. the command *Cleargrpahics* as *cg*, the command *right 30* as *rt 30*, the command *left 30* as *lt 30*, e.t.c.

2nd Tip: Executing the command:

```
ask "Turtle[primitives]
```

displays on the lower part of the 'Editor' all these primitives of Turtleworlds both in English and Greek.

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

3.2 Procedures – Sub-procedures

An important feature of Turtleworlds is that it enables the user to create his 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, Turtleworlds allows you to create your own additional words-commands besides the existing primitives and use them wherever and however you wish.

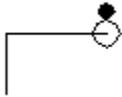
Every time you define a procedure-command it has the same status with the primitive commands. Thus, you can define a command by using another command you have already defined. In this way you create a limitless structure of procedures and sub-procedures.

3.3 Procedure construction

Let's say you have edited the following commands:

```
Forward 30
Right 90
Forward 50
Left 90
```

If you run these commands in this order, the following shape is created on the 'Canvas':



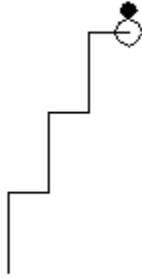
You can define a procedure that will run the above commands every time in the same order that it will be called. For this to be realized you give the command **"to"** and a name of your choice for the procedure. In a separate line, after the end of the series of commands that the new procedure will require,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

```
to stair
Forward 30
Left 90
Forward 50
Left 90
End
```

The word **"to"** is a primitive Turtleworlds 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 which 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 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 'Canvas'. Now that the procedure *stair* has been defined you can run the procedure as many times as you wish. For example, if you run 3 consecutive times the procedure the following shape will occur:



Advice: The procedures 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 is not a name that corresponds to a primitive Turtleworlds command. For example, the name *forward* is not acceptable.

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

3.3.1 Procedure inputs

The procedures can have inputs and outputs the same way as Turtleworlds commands. This is executed with the use of **variables**. By using a variable, the above procedure can be realized:

```
to stair :height
  Forward :height
  Forward 90
  Forward 50
  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 40*, a stair is created with a height of 40 steps.

Attention! Every time you use a variable in the code, the symbol **:** must precede its name. After **:** do **not** leave a space!

The names of the variables must also be a consecutive word. 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* so that a stair is created with 40 height and 80 width. 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.

3.4 Sub-procedures - Hyperprocedures

Turtleworlds allows for other procedures to be called within the procedures. Suppose the aforementioned procedure *stair : height : width* has been defined.

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

```
to stair : height : width
  stair 30 40
  stair : height : width
End
```

The new procedure *stair* 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* hyperprocedure.

Another example is the following:

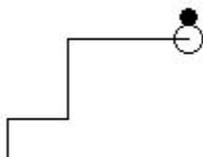
```
to doubleStairs : height : width
  stair : height : width
  stair : height*2 : width*2
End
```

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

The outcome from the execution of the above procedure as:

```
doubleStairs 20 30
```

is the following:



Advice: In a hyperprocedure, 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*.

4 Dynamic manipulation –Variation tools

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

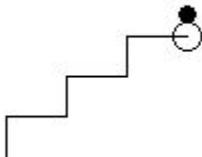
4.1 The Variation Tool

The 'Variation Tool' allows you to **dynamically** manipulate the variables of a function you have defined.

For example, 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 turtle draws the following shape:



If you move the mouse into the 'Canvas' you will observe a cross appearing. By using this cross you can left click on any part of the turtle's trace. Clicking on the trace indicated in the above example the 'Variation tool' (below the 'Canvas') acquires two sliders (Figure 2).

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

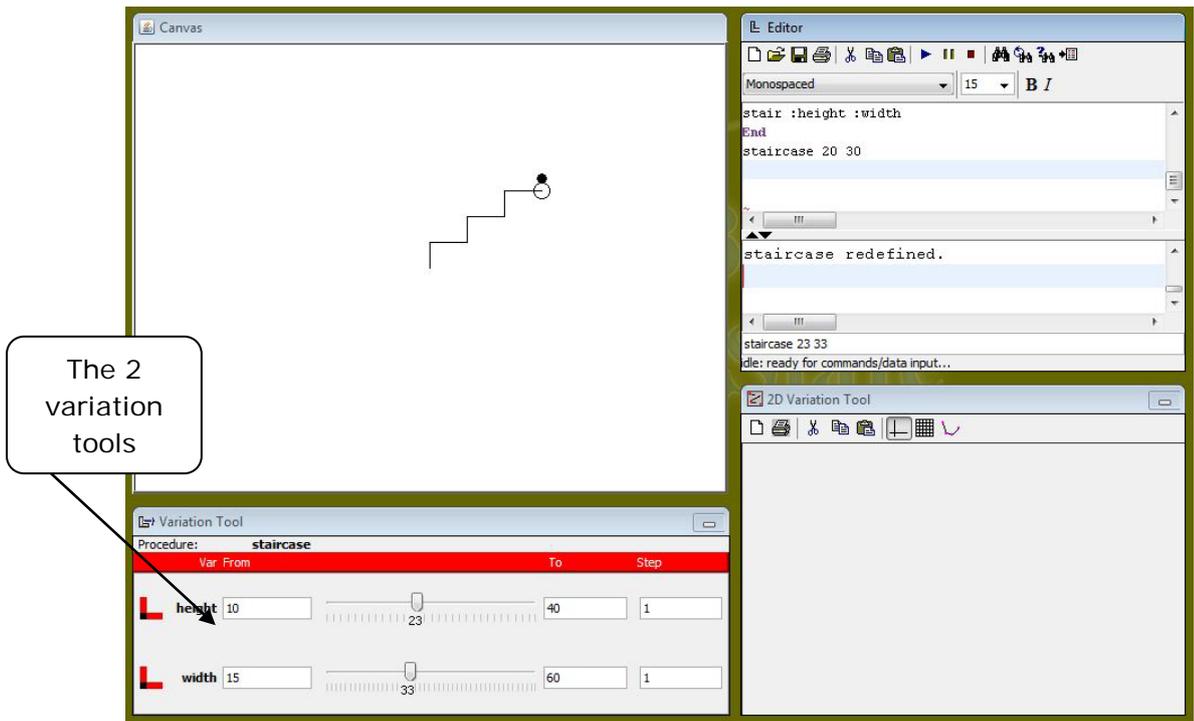


Figure 1: The variation tools corresponding to the height and width variables

From the left and 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 to 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.

4.2 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**.

Suppose you have defined the *staircase* procedure that was described above:

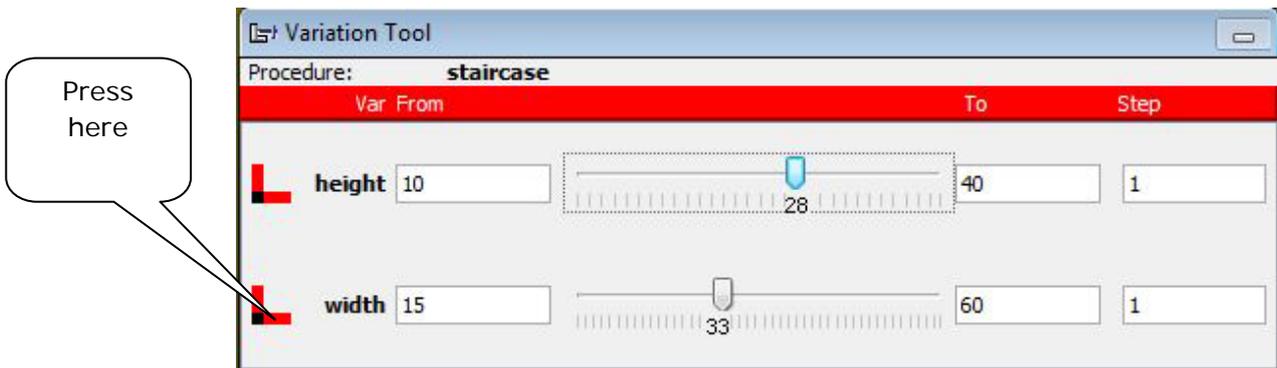
```
to staircase :height :width
  stair :height :width
  stair :height :width
  stair :height :width
```

End

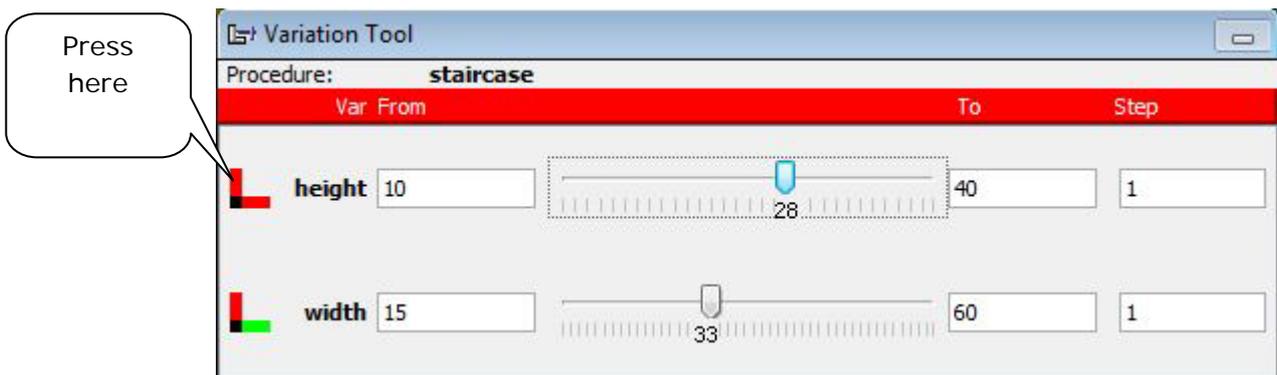
You have already activated the 'Variation tool'. On the left side of each slider there is a red right angle. By clicking on either side of the angle you can set this variable on the corresponding axis of the rectangular axis system, represented by the '2D Variation Tool'.

Suppose, for example, you want to create an axis system where the vertical axis will be the height and the horizontal axis will be the stair width. To do this, the following procedure must be followed:

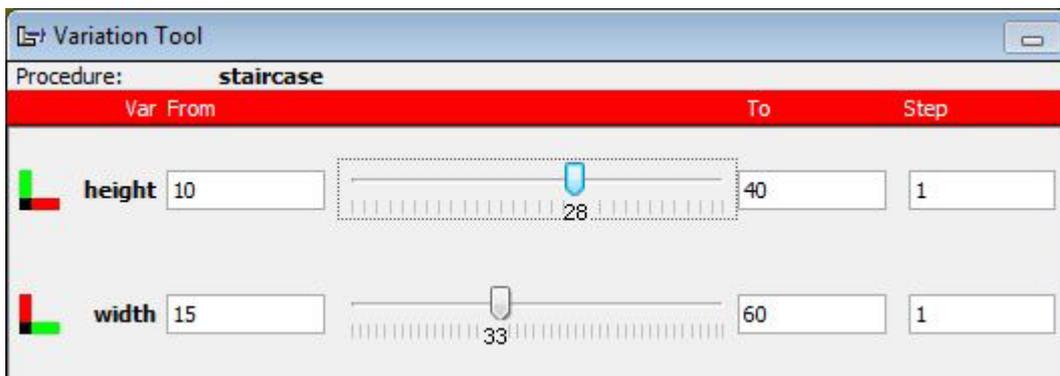
Allocation of the *width* variable to the horizontal axis.



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



The selected sides have changed colour 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. With the key pressed and by clicking on any point of the variation tool surface, a blue point is created with specific coordinates. The

height and width variable values have automatically taken the coordinate values of this point (Figure 3). Moving the blue point on the imaginary axis system both values of the height and width variation tools change automatically.

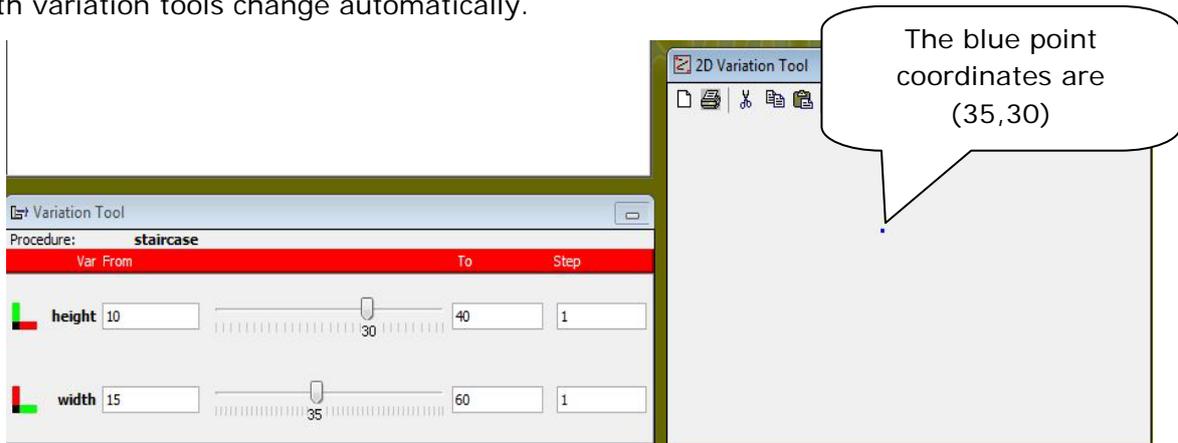


Figure 4: A point on the “2D Variation Tool” in Turtleworlds

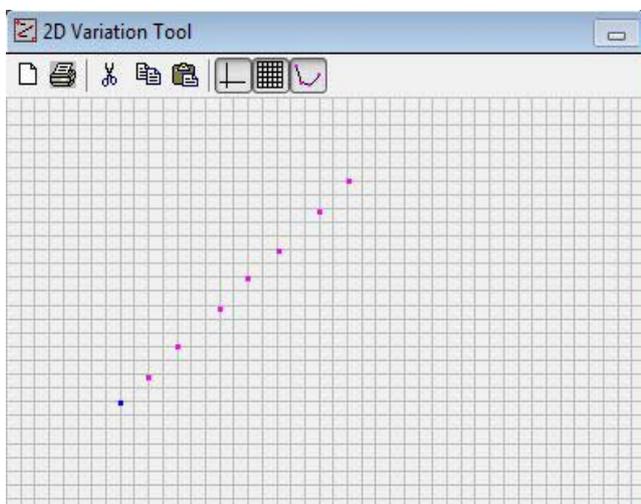
The button  of the ‘2D Variation Tool’ projects a grid on the axis system whereas the button  «clears» the variation tool from all the points that have been created.

Free dragging

If the button  is not pressed, 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 turtle.

4.3 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 3, in other words $height/width=3$. Try to place the blue point on the ‘2D Variation Tool’ in a position where the $height/width$ ratio values equals three. If you place some more blue points in different positions where the ratio of the two variables equals three, the following point set-up will occur:



Notice that the points make a straight line with a stable inclination. This line also constitutes the *height/width=3* 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 the variables correlate in order for certain conditions to be met.

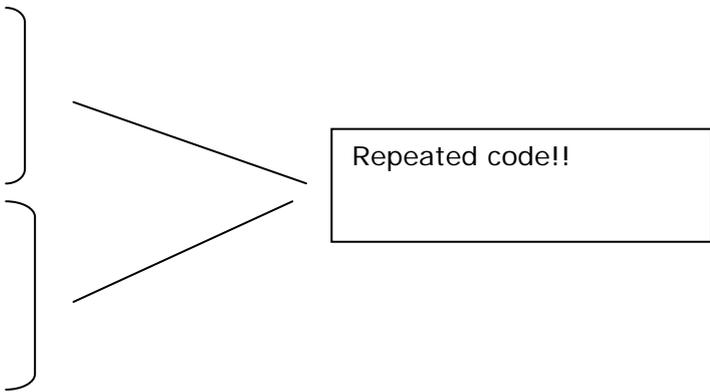
5 Repetition structure

Turtleworlds allows for 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. Suppose you want to define a procedure which results to the formation of a variable shaped parallelogram. One way to do this is the following:

```

to parallelogram :a :b :angle
  Forward :a
  Right :angle
  Forward :b
  Right 180-:angle
  Forward :a
  Right :angle
  Forward :b
  Right 180-:angle
End

```



This code defines the *parallelogram* procedure whose commands formulate a parallelogram. With the use of the repetition structure, the code to be repeated 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 Turtleworlds 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 turtle 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 with 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 Turtleworlds 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

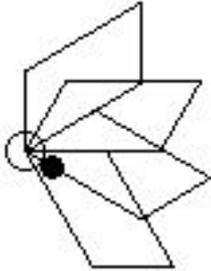
Repeat :times [parallelogram :a :b :angle right 30]

End

The *parallelograms* procedure repetitively calls the *parallelogram* procedure, 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 'Canvas':



6 Recursive procedures

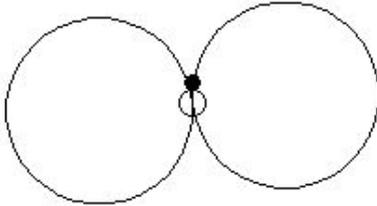
In the *Hyperprocedures* and *sub-procedures* section the way one procedure is called within another was described. Turtleworlds also allows for a procedure to call itself. This is called "recursion". Suppose you have defined the following procedure concerning the formation of a circle with radius r :

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

Suppose you have also defined the procedure:

```
for butterfly :r
  repeat 2 [circle :r rt 180]
end
```

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

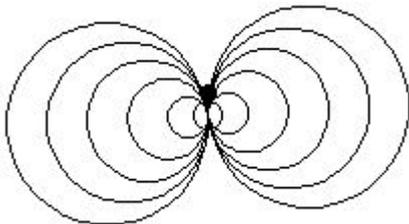


An example of the use of recursion in this procedure, is the following

```
to butterfly :n :r
  if :n < 1 [stop]
  repeat 2 [circle :r rt 180]
  butterfly :n-1 :r-10
end
```

Recursion: The procedure calls upon itself!

What is accomplished with the *butterfly* procedure above is the formation of two tangent circles and **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 turtle creates the following shape:



7 Control commands

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

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

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

Suppose the following procedure has been defined:

```
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 Turtleworlds 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 turtle will not do anything, since the condition is true.

But if it is executed as

```
stair 6 10
```

The turtle will create a stair.

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

Condition examples:

```
:height = 5
```

```
:height > :width
```

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

7.1 Control commands in recursion

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

```
to butterfly :n :r
  if :n < 1 [stop]
  repeat 2 [circle :r rt 180]
  butterfly :n-1 :r-10
end
```

If there wasn't for 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 discontinues. Thus, if you run the *butterfly* procedure as *butterfly 5 50*, what Turtleworlds does is the following:

Initially, it 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 cease. For the time being, this does not apply 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 cease after it has been run for 5 times (as many as the *n* initial value).

The following table shows in detail the *:r* 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	-

Appendix A – Tables of Commands

Table 1: Turtle control commands

Command	Number of Inputs	Number of Outputs	Description	Example
Cleargraphics or Cg	0	0	Clears the canvas and restores the turtle to its initial position	
Clean	0	0	Clears the canvas and lets the turtle in its position	
Home	0	0	Restores the turtle to its initial position without deleting what it has created	
Forward <i>number</i> or fd <i>number</i>	1	0	Moves the turtle forward as many steps as the <i>number</i> value	Forward 10 or fd 10
Back <i>number</i> or bk <i>number</i>	1	0	Moves the turtle backward as many steps as the <i>number</i> value	Back 10 or bl 10
Right <i>number</i> or rt <i>number</i>	1	0	Turns the turtle right by as many degrees as the <i>number</i> value	Right 90 or rt 90
Left <i>number</i> or lt <i>number</i>	1	0	Turns the turtle left by as many degrees as the <i>number</i> value	Left 90 or lt 90
Penup or pu	0	0	The turtle ceases to write on the Canvas	
Pendown or pd	0	0	The turtle writes on the Canvas	
Setpencolor[<i>value 1, value2, value3</i>]	1	0	Changes the turtle's colour based on certain colour codes	Setpencolor[255 0 0] Red colour (for more colours see below).
Setxy <i>position1 position2</i>	2	0	Places the turtle on the canvas coordinates (<i>position1, position2</i>)	Setxy 30 40
SETHEADING <i>degrees</i>	1	0	Orientates the turtle by as many degrees as the <i>degrees</i> input value	SETHEADING 90

Pos	0	2	Recalls the turtle's position coordinates	
XCOR	0	1	Recalls the turtle's abscissa position	
YCOR	0	1	Recalls the turtle's ordinate position	
canvaspagewidth	0	1	Recalls the Canvas' width	
canvaspageheight	0	1	Recalls the Canvas' height	

Basic colour codes RBG for the turtle's change of colour

Red	255	0	0
Green	0	255	0
Blue	0	0	255
Black	255	255	255
Yellow	255	255	0
Orange	255	128	0

To see all the colours visit the site http://www.rapidtables.com/web/color/RGB_Color.htm and choose the relevant codes.

Table 2: Mathematical function commands in Turtleworlds

Command	Number of Inputs	Number of Outputs	Description	Example	Output
Sum $a b$	2	1	Gives the sum of the two numbers set in its input, i.e. it performs $a+\beta$	Sum 3 5	8
Difference $a b$	2	1	Gives the difference of the two numbers set in its input, i.e. it performs $a-\beta$	Difference 8 3	5
Product $a b$	2	1	Gives the product of the two numbers set in its input, i.e. it performs $a*\beta$	Product 2 4	8
Quotient $a b$	2	1	Gives the quotient of the two numbers set in its input, i.e. it performs a/b	Quotient 6 3	2

Remainder $a b$	2	1	Gives the remainder of division of the two numbers set in its input.	Remainder 11 2	1
Root $number$	1	1	Gives the square root of the number set in its input	Root 36	6
Power $x n$	2	1	It raises the x number to the n power and returns the result. Thus, it is x^n	Power 2 4	16
Abs $number$	1	1	It returns the modulus of the number set in its input	Abs -10	10
Cos $degrees$	1	1	It returns the cosine of the angle set as an input	Cos 60	0.5
Sin $degrees$	1	1	It returns the sine of the angle set as an input	Sin 60	0.866
Tan $degrees$	1	1	It returns the tangent of the angle set as an input	Tan 180	0
Arccos $argument$	1	1	It returns the angle it calculates from the inverse cosine based on the argument set as an input	Arccos 0.5	60
Arcsin $argument$	1	1	It returns the angle it calculates from the inverse sine based on the argument set as an input	Arcsin 0.5	30
arctan $argument$	1	1	It returns the angle it calculates from the inverse tangent based on the argument set as an input	arctan 1	45
Exp $number$	1	1	It returns the exponential function with a base of e and as a power the number set in its input	Exp 1	2.718
Integer $number$	1	1	It returns the integer part of the number set as an input	Integer 2.8	2

Rounding number	1	1	It returns the rounding of the number set in its input	Rounding 2.3 Rounding 3.8	2 4
pi	0	1	It returns the π (3,14) number		3.14

Appendix B – Additional examples for the use of Turtleworlds

Example 1: Waves with staircases

Write and define the following procedure:

```
to stair_upper
  forward 2
  right 90
  forward 5
  left 90
end
```

Write and define the following procedure:

```
to stair_lower
  right 90
  forward 5
  right 90
  forward 2
  left 180
end
```

Write and define the following procedure:

```
to staircase_upper
  repeat 5[stair_upper]
end
```

Write and define the following procedure:

```
to staircase_lower
  repeat 5[stair_lower]
end
```

Write and define the following procedure:

```
to staircase_upright
  staircase_upper
  staircase_lower
end
```

Write and define the following procedure:

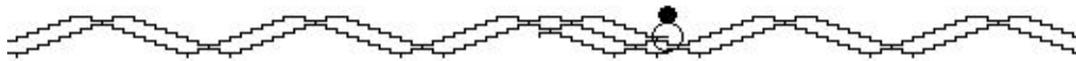
```
to staircase_reverse
  staircase_lower
  staircase_upper
```

end

Write and define the following procedure:

```
to waves
  Repeat 11[staircase_upright staircase_reverse]
end
```

When the last procedure **waves** is run, it calls the **staircase_upright** sub-procedure as well as the **staircase_reverse** sub-procedure which in turn call the **staircase_upper** sub-procedure and the **staircase_lower** sub-procedure with the latter to call next the **stair_upper** and **stair_lower** sub-procedures, respectively. As a consequence, the turtle creates on the 'Canvas' area something that looks like waves, as seen below:



Example 2: Diagonal formation of a canonical polygon with recursion

In the following example the use of recursion is demonstrated for the construction of all the diagonals of a n-gon.

The defined procedures are the following:

```
to diagonals :n :s
  repeat :n[
    design_diagonals :n :s 2
    lt ((:n-3)*(180/:n))
    fd :s
    rt (360/:n)
  ]
end

to design_diagonals :n :s :k
  if :k > (:n-2) [stop]
  rt 180/:n
  fd length_diagonal :n :s :k
  bk length_diagonal :n :s :k
  design_diagonals :n :s :k+1
end

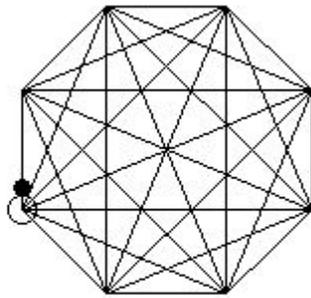
to diameter :n :s
```

```
output 2*((:s*sin (180*(:n-2)/(2*:n)))/sin (360/:n))
end
```

```
to length_diagonal :n :s :k
output (diameter :n :s)*sin (180*:k/:n)
end
```

The main procedure is the **diagonals :n :s**, which takes as inputs the number of $:n$ sides of the polygon and the $:s$ length of the side. Then for each angle of the polygon the procedure **design_diagonals** is called which recursively designs the angle's diagonals. In the following figure an example of an 8-gon with a length side of 60 is displayed.

The execution is realized with the *diagonals 8 60* command.



Example 3: Designing the “Rosette” shape by applying recursion

The rosette design is a complex one constructed on the basis of geometric rules and functions. First, a number of concentric circles are designed with a radius of twice, three times and so on, the radius of the inner circle. Next the inner circle is divided into equal arcs. The cycle chords of these arcs constitute the isosceles triangles bases whose vertexes are in the next circle. The vertex points of the first line triangles constitute the base points of the second line triangles. You continue this way until you complete the last line of triangles.

The functions defined in Turtleworlds are the following:

```
fto rhombus :a :b
make "θ arccos(:b/(2*:a))
lt :θ
repeat 2 [
fd :a rt 2*:θ fd :a rt 180-2*:θ]
rt :θ
end
```

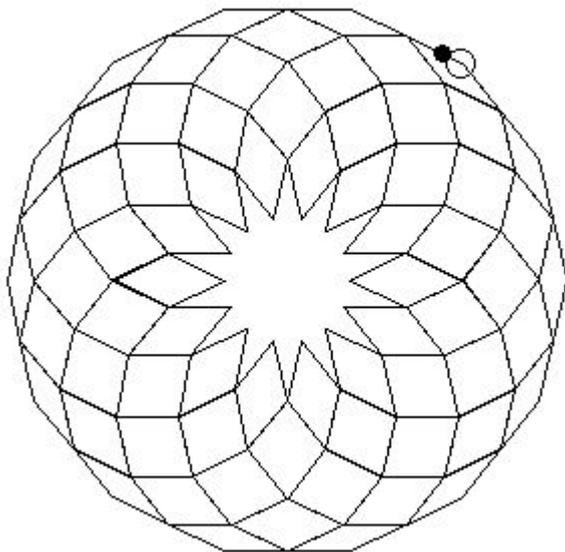
```
to rosette :n :a :b
```

```
make "θ arccos(:b/(2*:a))
if :θ<90/:n [stop]
repeat :n [
  rhombus :a :b
  pu fd :b pd
  lt 360/:n]
rt :θ fd :a
lt 2*:θ rt :θ-180/:n
rosette :n :a 2*:a*cos(:θ-180/:n)
end
```

The basic procedure is `rosette :n :a :b` and its execution as:

```
rosette 14 31 27
```

has the following outcome:



Notice:

The present "Manual" has been composed by researchers in the Educational Technology Lab in cooperation with teachers that have extensively used "Turtleworlds" in the classroom.