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

"D-stage" Manual

Table of contents

1. Introduction 2
1.1 The D-stage kit 2
2. Description of D-stage 3
2.1 D-stage software environment 3
2.2 D-stage category description of components 4
3. D-stage Structure and function 7
3.1. Extension: Graph construction 12
4. Component connectivity 13
4.1. Component connectivity through «plug editor» 13
4.2. Component connectivity through Logo scripts 14
5. Brief presentation and Logo code analysis 17

1. Introduction

This manual is addressed to every student, teacher or researcher who wishes to utilize «D-stage» software to construct educational software, without deep programming knowledge.

1.1 The D-stage kit

D-stage software (or "Dyna-stage") (kit), is a microworld, constructed with E-slate authoring system. D-stage microworld is a model for constructing new microworlds for Physics with an emphasis on the Mechanics field. The initial objective of the microworld is the study of projectile motion.

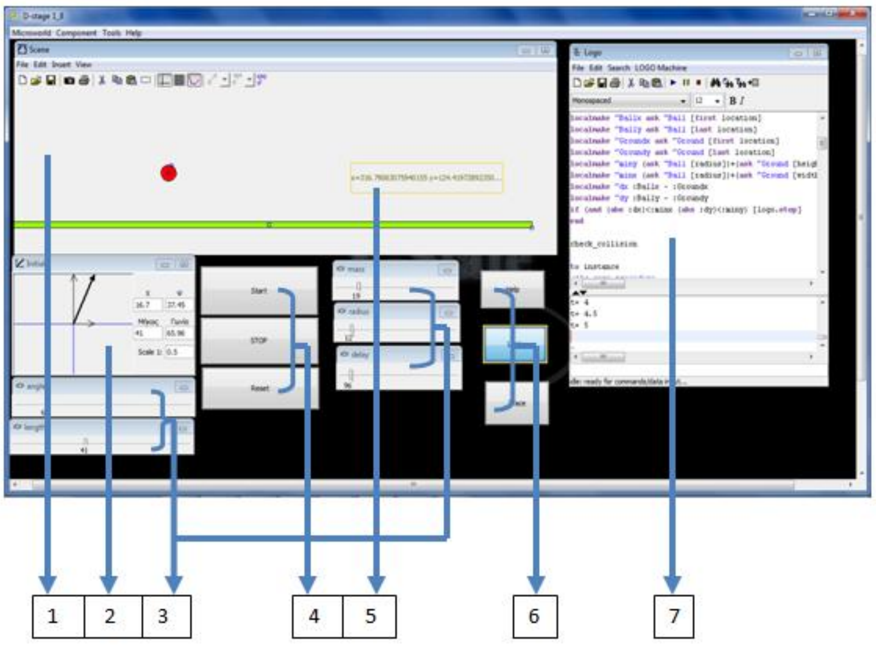
The study and simulation of an abundance of natural phenomena becomes possible by appropriately adjusting the components of the microworld and by adding new functionalities to them.

2. Description of D-stage

2.1 D-stage software environment

D-stage software environment consists of 19 construction units, each of which performs certain functions. These construction units are called "components" and are provided as a library of pre-fabricated computational objects, especially designed for educational use. D-stage components can be “plugged” together or can be programmed through the use of the “Editor” command component, so as to attain functionality.

The figure below depicts D-stage components grouped into categories ("forms") with identical functionality:



```
localmake "Ballx ask "Ball [first location]
localmake "Bally ask "Ball [last location]
localmake "Groundx ask "Ground [first location]
localmake "Groundy ask "Ground [last location]
localmake "miny (ask "Ball [radius])+(ask "Ground [heig
localmake "minx (ask "Ball [radius])+(ask "Ground [widt
localmake "dx :Ballx - :Groundx
localmake "dy :Bally - :Groundy
if (and (abs :dx)<:minx (abs :dy)<:miny) [logo.stop]
end

check_collision

to instance
```

```
t= 4
t= 4.5
t= 5
```

Figure: 1 D-stage component categories

2.2 D-stage category description of components

1. **Stage:** The Stage component provides an environment for creating "scenes" of spatial configurations of objects such as boxes, balls, springs, ropes, etc. The physical characteristics of these objects, such as size, color and position, are determined either by direct manipulation (by using the mouse) or via the scripting mechanism (commands written in the “Editor”). In this way, simulations and experiments in Physics can be performed on the Stage component. Logo scripting orchestrates their behavior (e.g. the motion of bodies in relation to time).

2. **Vector:** This component provides the ability to define and manage vector sizes in a straightforward manner. The vector value is determined either by the numerical components, or by dragging the mouse to the desired end of the vector on the graphical display.

3. **Slider:** The slider component graphically changes the values of a numerical variable. By plugging the slider together with other components, it provides these values for further use and processing.

4. **Button:** The button component is a graphical object. By clicking the button a predetermined operation can be performed.

5. **Label:** This component is used only for text display

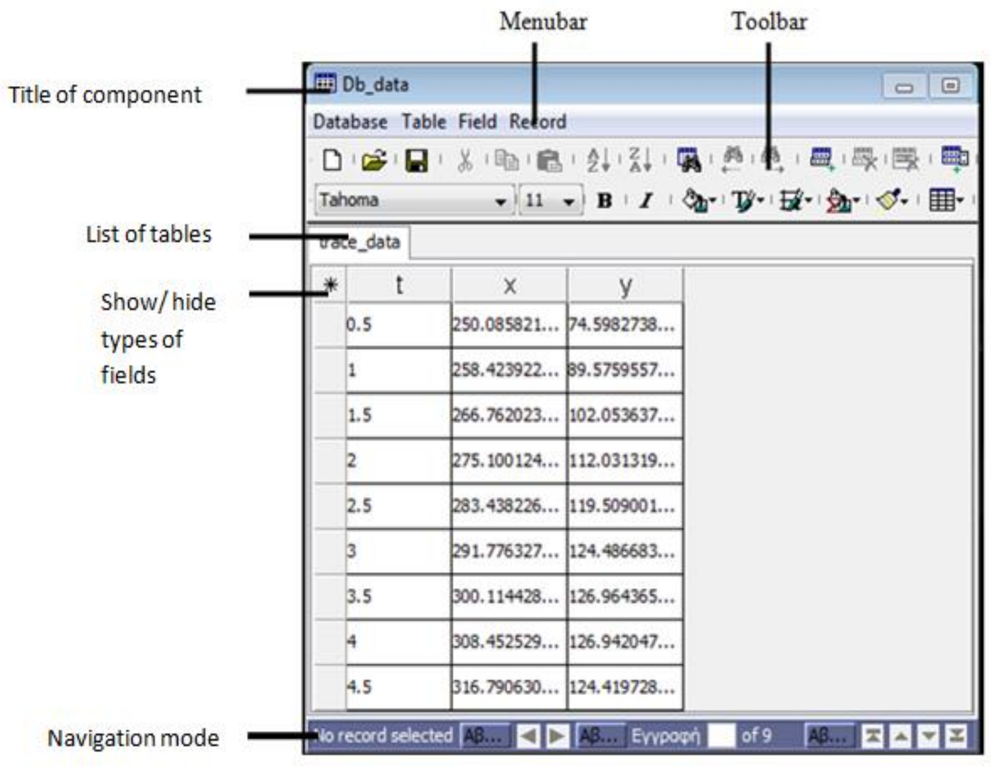
6. **Toggle Button:** This component is like the "Button" component but with two positions: on and off. Tbutton, is used in order to induce switching of situations (such as a switch on / off)

7. **Logo:** This is the Editor component which provides the means for editing and executing commands with the use of programming language LOGO. Logo scripting orchestrates the microworld's funcionality.

When opening the D-stage software environment the following components do not appear but they are activated when an “event” is fired for example after pressing a button:

1) **Text:** Text component provides basic editing text functions and formatted text display functions. Text component can accept text from other components (through corresponding plug) for display or for processing.

2) **Database Editor:** The Database Editor component enables the creation and general management (storage, sorting, categorization, etc.) of data in the form of one or more tables. Data can be created at the outset, or imported from other connected components, or even incorporated by other applications (eg Microsoft Excel, Microsoft Access) through intermediate files.



*	t	x	y
	0.5	250.085821...	74.5982738...
	1	258.423922...	89.5759557...
	1.5	266.762023...	102.053637...
	2	275.100124...	112.031319...
	2.5	283.438226...	119.509001...
	3	291.776327...	124.486683...
	3.5	300.114428...	126.964365...
	4	308.452529...	126.942047...
	4.5	316.790630...	124.419728...

Figure 2: database editor component

3) **Canvas:** The canvas component is a surface where a series of different design functions separated into "painting pages" can happen. Designs (eg graphs) created on the canvas can be done either by using pre-fabricated design library or dynamically with the use of connected ”turtle” which is a graphical object that leaves a trace on the canvas depending on the scripts written in the Editor .

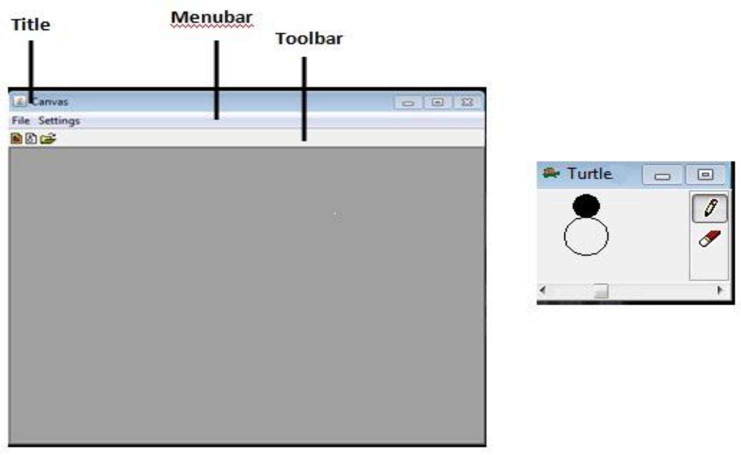


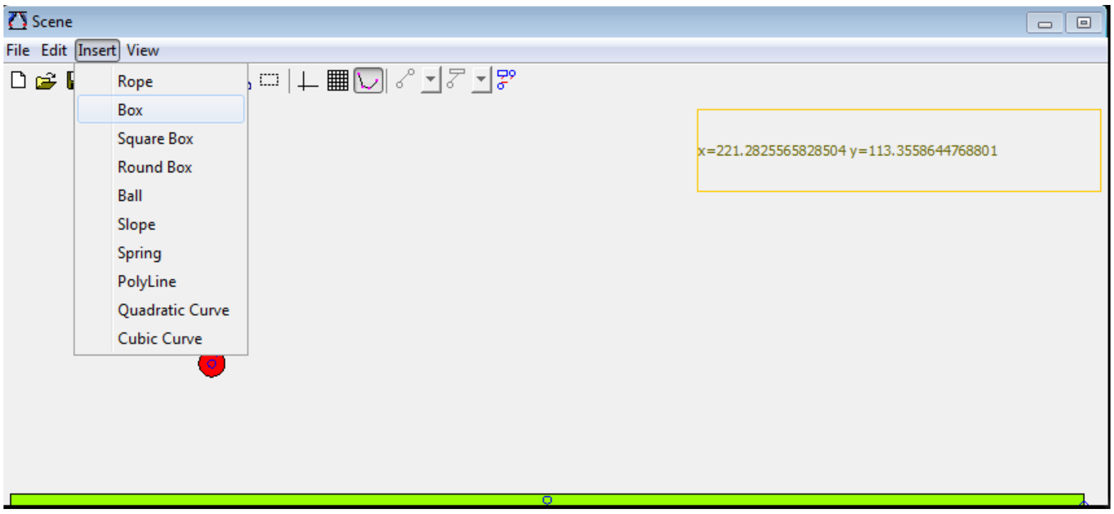
Figure 3: Canvas and Turtle components

4) **Turtle:** This component depicts the classic “LOGO-turtle” that leaves its trace on the Canvas component (when these two are connected together).

The number of components used by a microworld can be seen by selecting components through the [component] > [components menu]. A component’s properties may be manipulated through the Property Editor by selecting [Tools] > [Component editor]. By using the option [Tools]> [Task bar] all units of the microworld appear in the task bar.

3. D-stage Structure and function

The basic element of the D-stage is the stage («Scene»). At the stage component appears the ball and the base (grass) on which the ball is going to land after executing projectile motion. These two objects are inserted on stage through the menu [Insert]> [Box] of the component. The manipulation of properties (color, position within the stage, mass, etc.) can be possible by right-clicking on the item "Box" and by selecting "Properties". By following the same procedure the ball is placed into the stage and its properties are manipulated. In the same way, a required number of objects can be placed and manipulated in the stage.



x=221.2825565828504 y=113.3558644768801

Figure 4: insert object

At the top right of the surface of the stage, a component label has been placed. This label displays (via the command editor) the x and y coordinates of the ball at the stage. The origin (O) can be changed by selecting [View]> [Axes] and by dragging with the left mouse button.

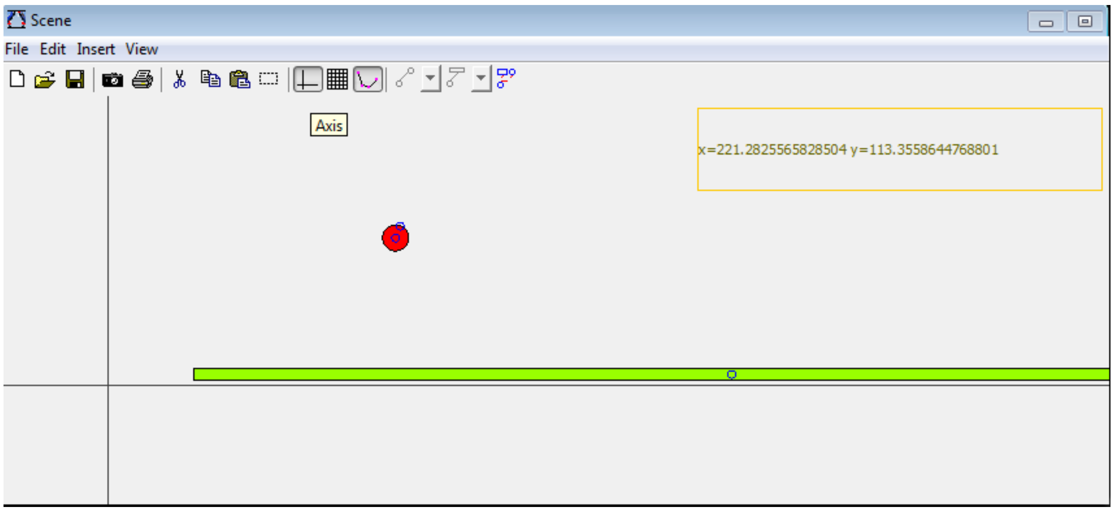


Figure 5: Change of the origin (O)

The determination of ball characteristics and movement is done by the sliders: "mass", radius "," delay ".

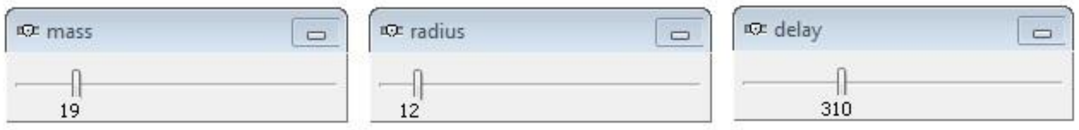
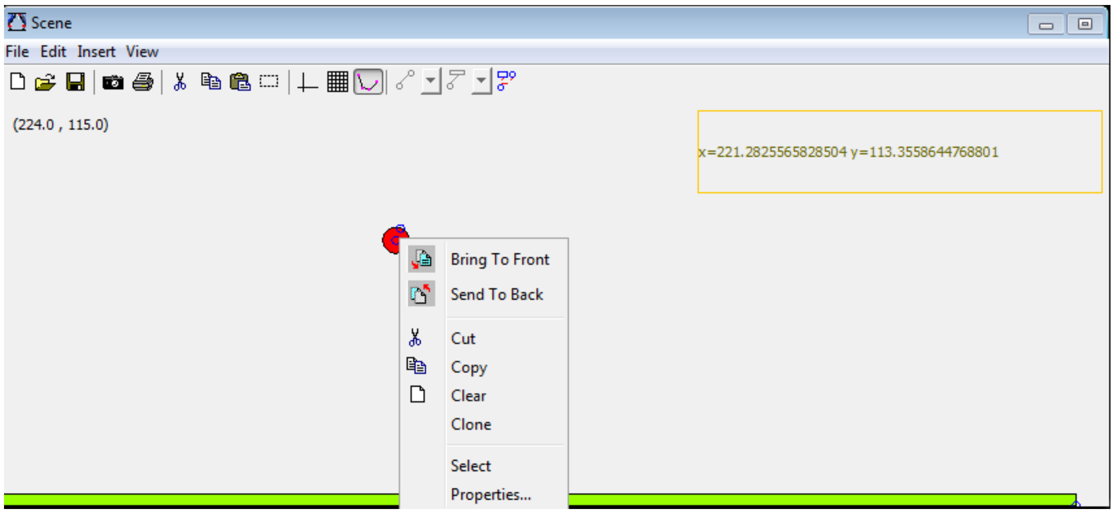


Figure 6: The sliders

The slider "mass", regulates the mass of the ball and the slider "radius" sets the radius of the ball. The slider "delay" delays the progress of the simulation so as to make the movement of the ball understandable

Note: The mass of the ball is not used in D-stage calculations. The slider “mass” remains for implementation of alternative scenarios on the projectile motion

The initial characteristics of an object (eg the ball) can also be changed through their stage properties by right clicking on the object.

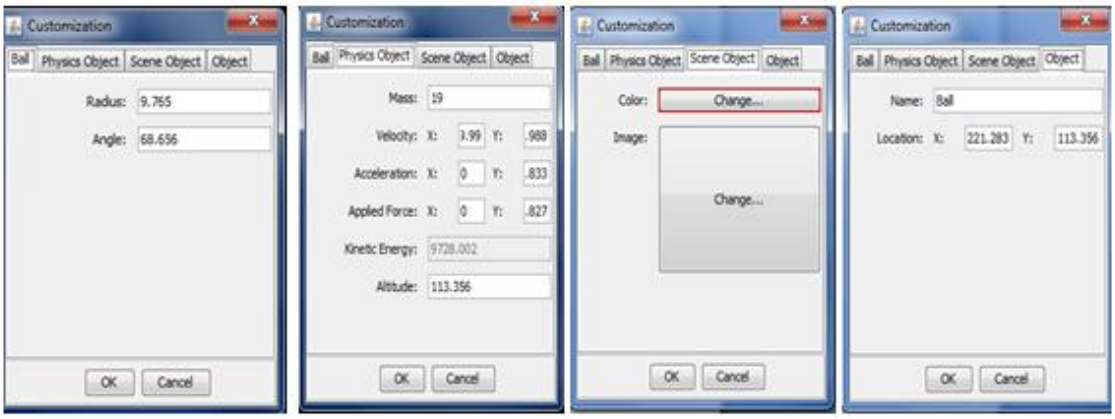


(224.0 , 115.0)

x=221.2825565828504 y=113.3558644768801

Figure7: Options by right-clicking on the object

A number of properties can be changed, from the name and shape of the background object, to its physical characteristics, such as its radius, its mass, and also data that characterize its position and movement in space, such as the force exerted, its acceleration, its velocity.



Radius: 9.765
Angle: 68.656

Mass: 19
Velocity: X: 3.99 Y: .988
Acceleration: X: 0 Y: .833
Applied Force: X: 0 Y: .827
Kinetic Energy: 9728.002
Altitude: 113.356

Color:
Image:

Name: Ball
Location: X: 221.283 Y: 113.356

Figure 8: object properties

The «Initial» component is a "Vector" component. It depicts the vector of the initial velocity of the ball, i.e. its measure and its angle of projection. The change of initial velocity is possible with the direct change of components' data

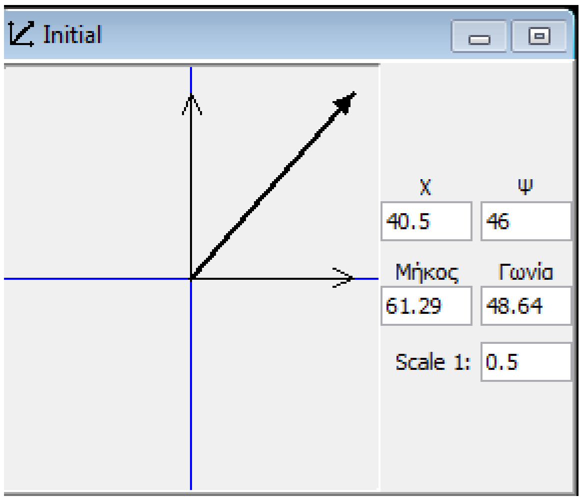


Figure 9: the vector of initial velocity

The initial velocity of the projectile motion can be alternatively determined by the sliders «Angle», «Length» by varying the angle of projection and measure the velocity respectively.

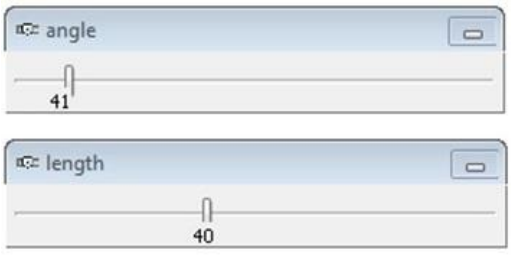
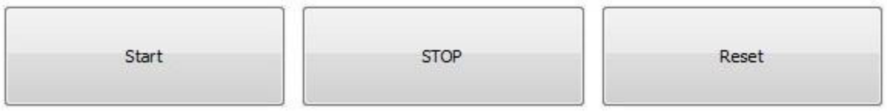


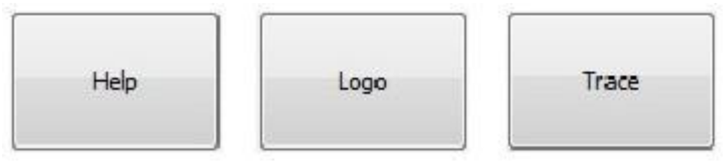
Figure 10: The sliders Angle, Length

The buttons «Start», «Stop» and «Reset» are the control buttons of the motion simulation.

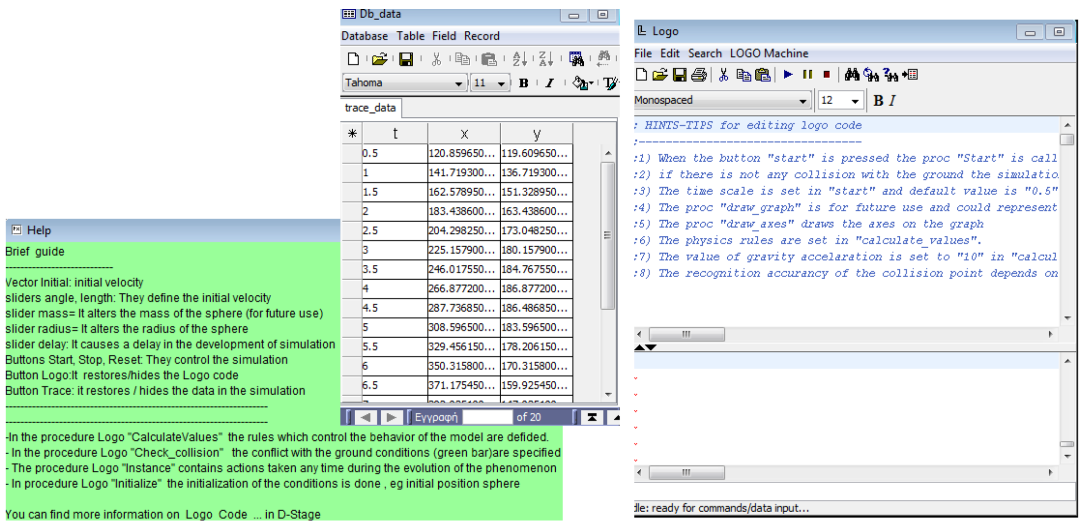


The simulation starts by clicking on the «Start» button and stops by clicking on the «Stop» button. The ball returns to its initial position by clicking on the «Reset» button (without also resetting the initial values of the vectors' velocity). If «Stop» button is pressed, during the course of the simulation the initial conditions can be redefined and the ball (with the new values) can continue to move.

Finally, on D-stage software environment, Tbuttons «Logo», «Trace», «Help» restore and hide the respective components.



Script written in the «Logo» component triggers component functionality of the microworld. The «Trace» button restores and hides a database which is filled with data movement of the ball, on each run of the simulation i.e. with the points (x, y) of which the ball passes as the time (t) elapses. These data can be used for further study-development of phenomenon. Finally, the «Help» button displays a brief guide to the Microworld.



t	x	y
0.5	120.859650...	119.609650...
1	141.719300...	136.719300...
1.5	162.578950...	151.328950...
2	183.438600...	163.438600...
2.5	204.298250...	173.048250...
3	225.157900...	180.157900...
3.5	246.017550...	184.767550...
4	266.877200...	186.877200...
4.5	287.736850...	186.486850...
5	308.596500...	183.596500...
5.5	329.456150...	178.206150...
6	350.315800...	170.315800...
6.5	371.175450...	159.925450...

Brief guide

Vector Initial: initial velocity
sliders angle, length: They define the initial velocity
slider mass= It alters the mass of the sphere (for future use)
slider radius= It alters the radius of the sphere
slider delay: It causes a delay in the development of simulation
Buttons Start, Stop, Reset: They control the simulation
Button Logo:It restores/hides the Logo code
Button Trace: it restores / hides the data in the simulation

-In the procedure Logo "CalculateValues" the rules which control the behavior of the model are defided.
- In the procedure Logo "Check_collision" the conflict with the ground conditions (green bar)are specified
- The procedure Logo "Instance" contains actions taken any time during the evolution of the phenomenon
- In procedure Logo "Initialize" the initialization of the conditions is done , eg initial position sphere

You can find more information on Logo Code ... in D-Stage

```
; HINTS-TIPS for editing logo code
;---------------------------------
;1) When the button "start" is pressed the proc "Start" is call
;2) if there is not any collision with the ground the simulatio
;3) The time scale is set in "start" and default value is "0.5"
;4) The proc "draw_graph" is for future use and could represent
;5) The proc "draw_axes" draws the axes on the graph
;6) The physics rules are set in "calculate_values".
;7) The value of gravity accelaration is set to "10" in "calcul
;8) The recognition accurancy of the collision point depends on
```

dle: ready for commands/data input...

Figure 11: The components displayed by the corresponding buttons

3.1. Extension: Graph construction

On the D-stage kit as the simulation progresses there is provision for a dynamic graph creation of a size. For this purpose a "Canvas" component and an interconnected "turtle" have been incorporated. The turtle through scripts written in the LOGO editor can “etch" the axes and then "paint" the intermediate points of the graph in the course of the phenomenon.

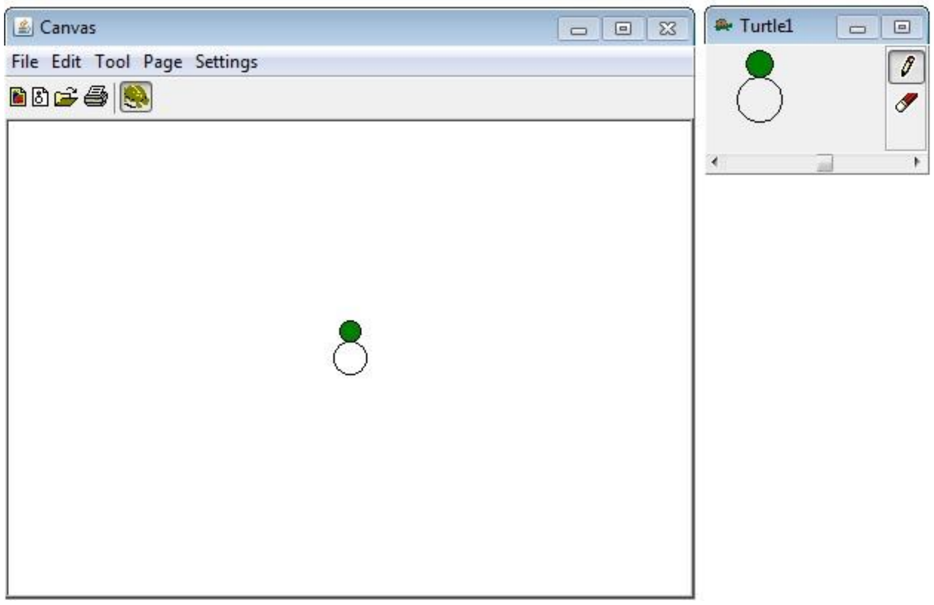


Figure 12: The components Canvas and Turtle

4. Component connectivity

The interconnection of components in order to attain functionality can be done in two ways (a) through the” plug editor” and (b) through scripts written in the LOGO editor.

4.1. Component connectivity through «plug editor»

The components are able to exchange data between them through various types of plugs and thus they have a function which is easy and "automatic".

The projection of the existing links and the creation of new ones can be done by selecting the [Tools] > [Plug Editor]

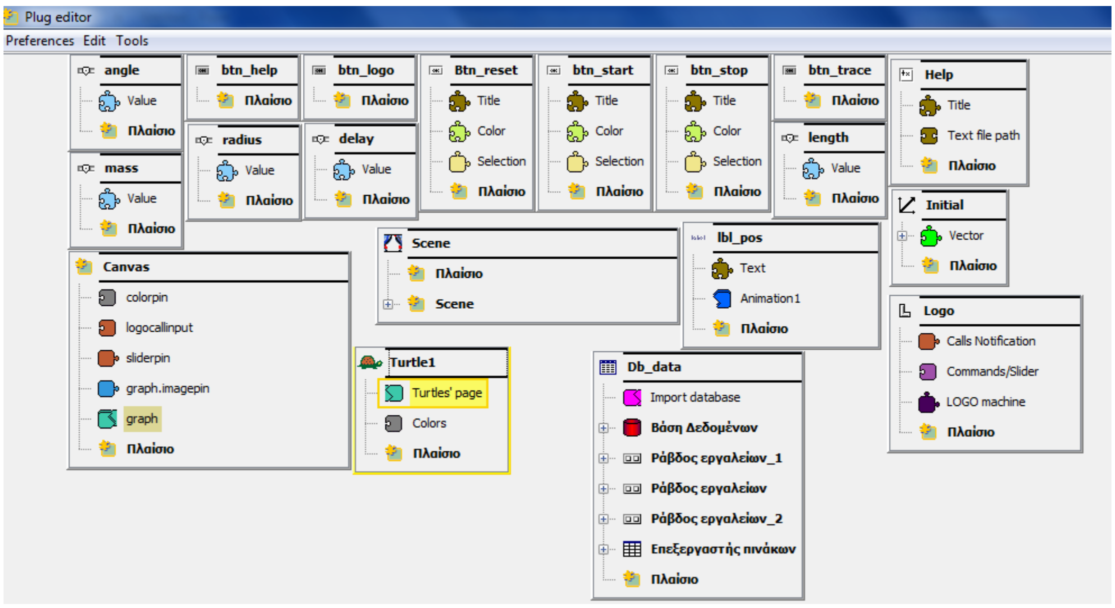


Figure13: plug editor

Each component has its corresponding plugs which describe the component properties that can be used in connection with other components. A typical example of this is the connection of turtle «Turtle1» with the canvas «Canvas» so the turtle can “draw” on the canvas. By clicking on a component plug all corresponding plugs from other components automatically highlighted in order to establish inter- component connections. The connection is established by dragging a source Plug onto a target Plug.

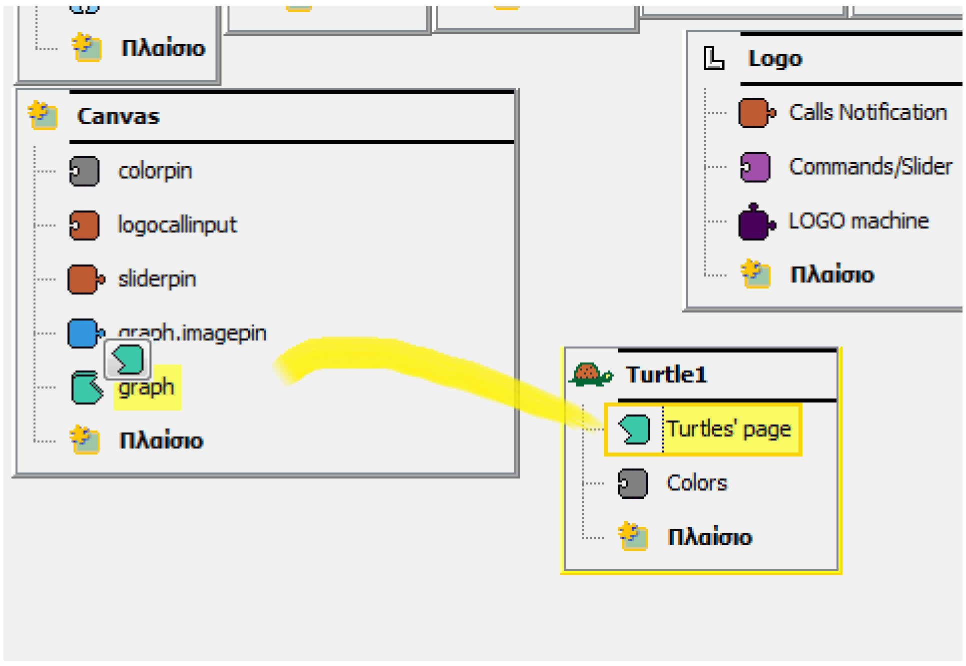


Figure14: component connectivity

4.2. Component connectivity through Logo scripts

The full configuration and adjustment of a microword function can be achieved by inputting Logo scripts in the respective component

Example: In order to place the ball in a specific location of the stage by pressing a button, it is necessary to:

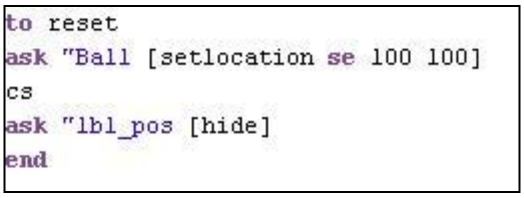
A) Define a procedure (i.e. a small script) in the LOGO Editor.

B) Input the procedure name in the appropriate "event" which is activated by pressing the button

In this way the button "is connecting" with the corresponding action which is specified in the LOGO script

Specifically: When “Reset” button is pressed, the ball moves to its initial position:

The following code is inputted in the Logo Editor:



```
to reset
ask "Ball [setlocation se 100 100]
cs
ask "lbl_pos [hide]
end
```

The procedure (which always starting with «to» and ending with «end»)is called “reset” and in order to be valid is needed to be "set" by highlighting and pressing [INSERT] According to the included commands the ball «Ball» is placed in [100.100] position and the label which depicts the position is hidden.

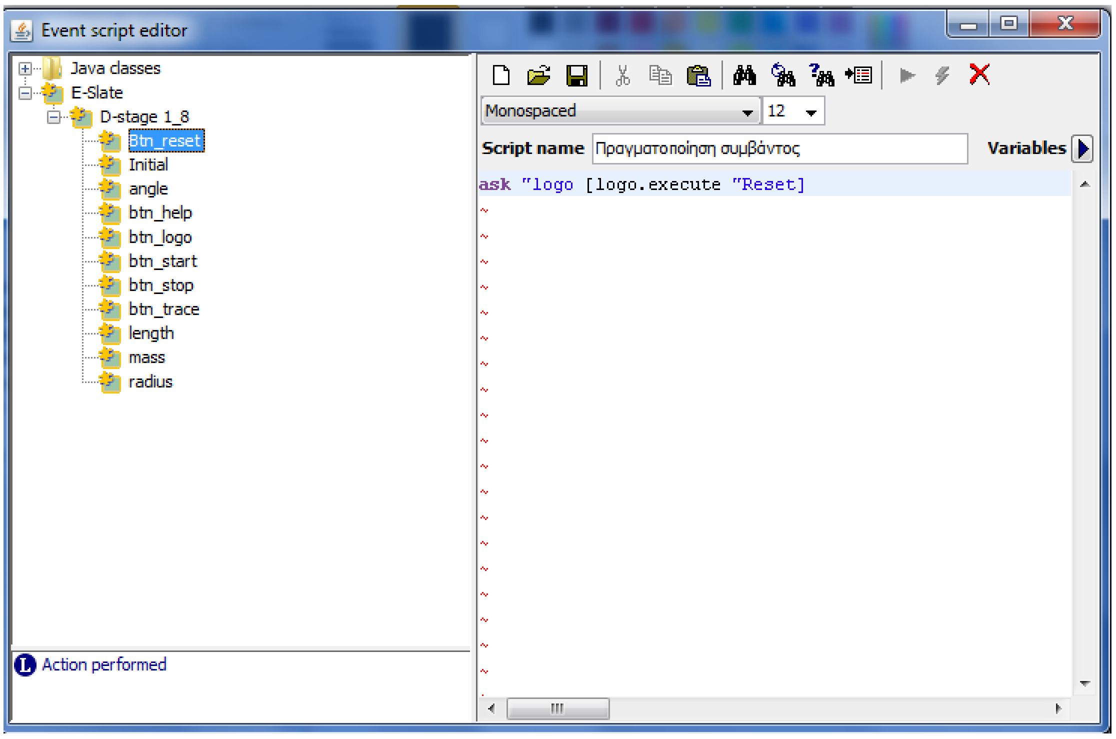
In order to make the connection with the «Reset» button component the event "Action performed" is needed to be defined in the following way:

A)Select Component management ([Tools]>[Component editor])

B) From the window that appears, select from the popup list the component «Btn_reset» which is a button component

C) From the tab «Events» select the «L» button which is next to “Action performed”

D) In the window “event script editor" which appears, enter the command: ask" Logo [Logo.execute "reset] and then select the button with the lightning symbol. With this command when the specified event is activated (by clicking the mouse) automatically runs the procedure called “reset” which is written in LOGO.



Script name: Πραγματοποίηση συμβάντος

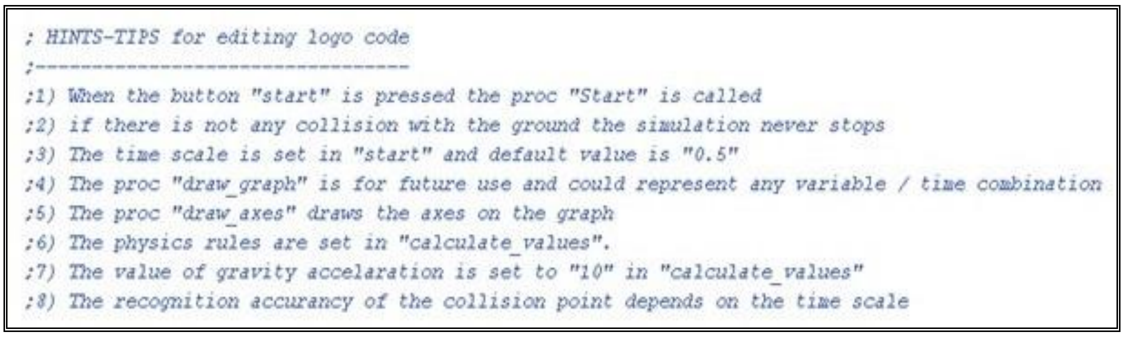
```
ask "logo [logo.execute "Reset]
```

Figure 15: Command input on "event"

5. Brief presentation and Logo code analysis

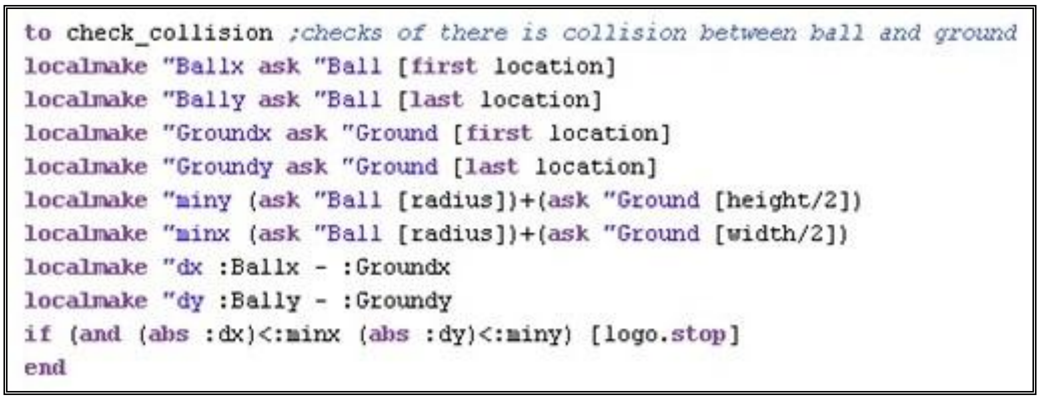
Below are the procedures by which the microworld attain funcionalities and can be found in the Logo Editor.

Some instructions on the code written by the constructor of the microworld are initially appeared. If we enter in front of any command the Greek question mark (;), the command is disabled and is considered as a comment.



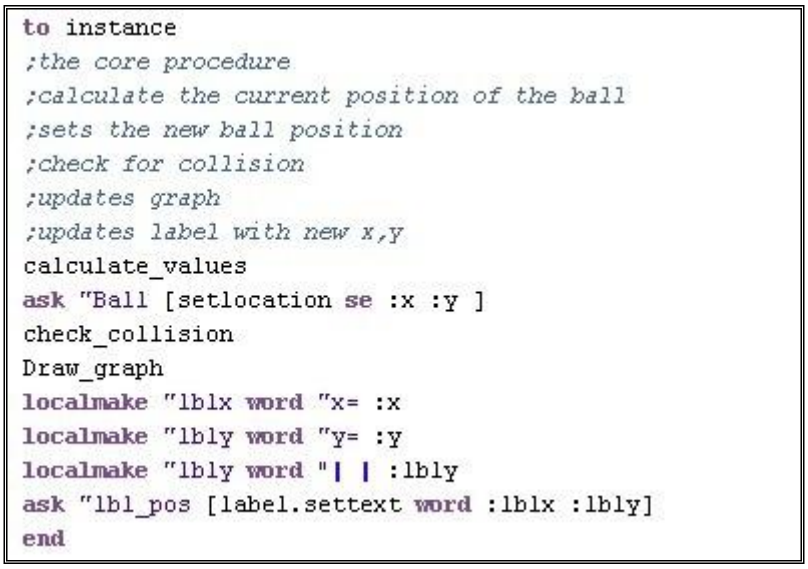
```
; HINTS-TIPS for editing logo code
;-----------------------------------
;1) When the button "start" is pressed the proc "Start" is called
;2) if there is not any collision with the ground the simulation never stops
;3) The time scale is set in "start" and default value is "0.5"
;4) The proc "draw_graph" is for future use and could represent any variable / time combination
;5) The proc "draw_axes" draws the axes on the graph
;6) The physics rules are set in "calculate_values".
;7) The value of gravity accelaration is set to "10" in "calculate_values"
;8) The recognition accurancy of the collision point depends on the time scale
```

Then the code sets the conditions under which the movement of the ball stops. The position of the ball relative to the position of the level is checked. If the ball has reached the level of the floor, then asks from the simulation to stop.



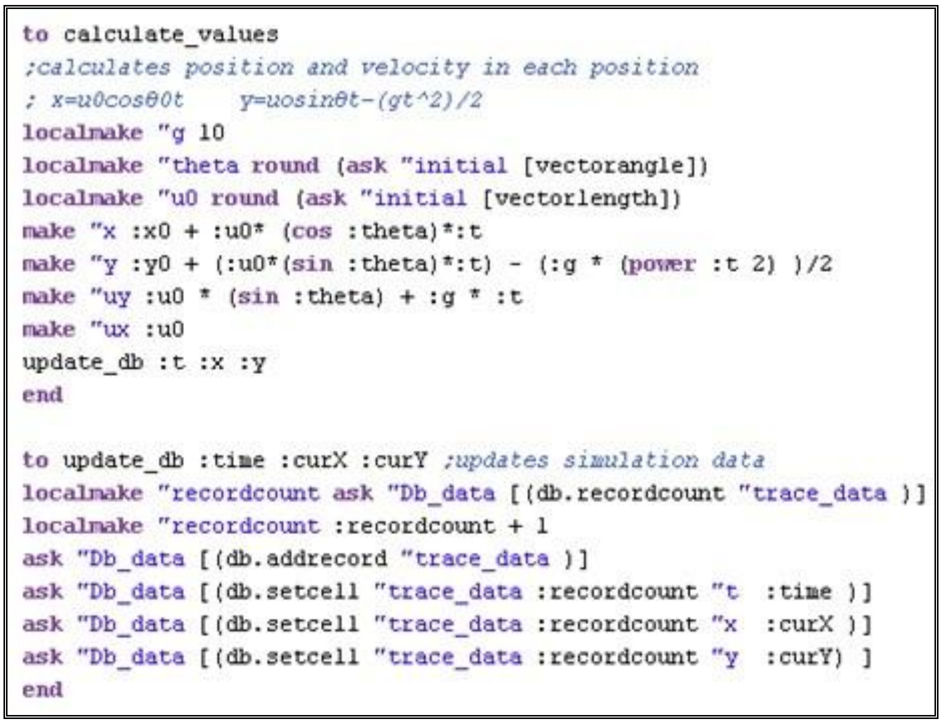
```
to check_collision ;checks of there is collision between ball and ground
localmake "Ballx ask "Ball [first location]
localmake "Bally ask "Ball [last location]
localmake "Groundx ask "Ground [first location]
localmake "Groundy ask "Ground [last location]
localmake "miny (ask "Ball [radius])+(ask "Ground [height/2])
localmake "minx (ask "Ball [radius])+(ask "Ground [width/2])
localmake "dx :Ballx - :Groundx
localmake "dy :Bally - :Groundy
if (and (abs :dx)<:minx (abs :dy)<:miny) [logo.stop]
end
```

The «instance» procedure is the "central" simulation procedure which performs all actions required in each moment. So, this procedure calculates the position of the ball (through a separate procedure) sets the new position, checks if the ball touches the ground, designs graph (for future use) and renews the label with the ball coordinates.



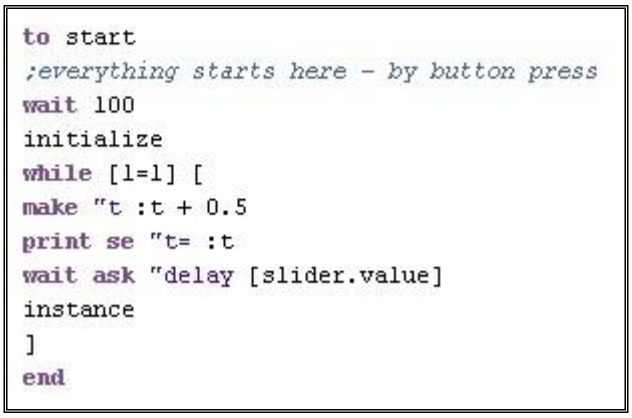
```
to instance
;the core procedure
;calculate the current position of the ball
;sets the new ball position
;check for collision
;updates graph
;updates label with new x,y
calculate_values
ask "Ball [setlocation se :x :y ]
check_collision
Draw_graph
localmake "lblx word "x= :x
localmake "lbly word "y= :y
localmake "lbly word "| | :lbly
ask "lbl_pos [label.settext word :lblx :lbly]
end
```

For the ease of maintenance, the calculation of the ball position is in a different procedure. Here is the 'part' of Physics, where the relations for the position and velocity in both axes are established. The necessary constants (g = 10m/s2)are defined and the initial values are recovered: the angle θ, the initial velocity. Database updates with the motion data in a separate procedure («update_db»).



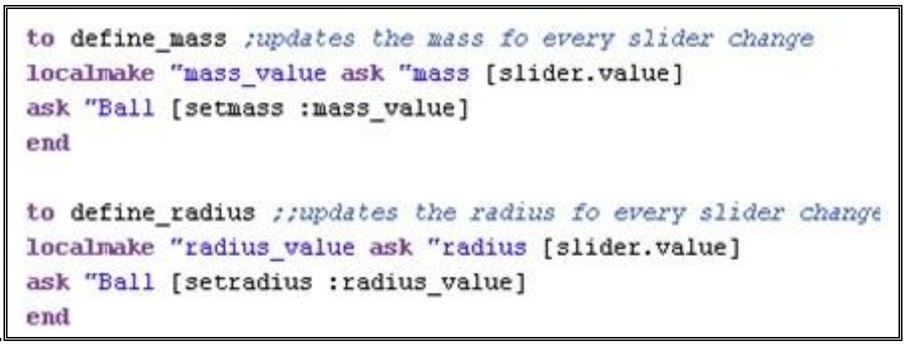
```
to calculate_values
;calculates position and velocity in each position
; x=u0cosθ0t     y=uosinθt-(gt^2)/2
localmake "g 10
localmake "theta round (ask "initial [vectorangle])
localmake "u0 round (ask "initial [vectorlength])
make "x :x0 + :u0* (cos :theta)*:t
make "y :y0 + (:u0*(sin :theta)*:t) - (:g * (power :t 2) )/2
make "uy :u0 * (sin :theta) + :g * :t
make "ux :u0
update_db :t :x :y
end

to update_db :time :curX :curY ;updates simulation data
localmake "recordcount ask "Db_data [(db.recordcount "trace_data )]
localmake "recordcount :recordcount + 1
ask "Db_data [(db.addrecord "trace_data )]
ask "Db_data [(db.setcell "trace_data :recordcount "t  :time )]
ask "Db_data [(db.setcell "trace_data :recordcount "x  :curX )]
ask "Db_data [(db.setcell "trace_data :recordcount "y  :curY) ]
end
```

The simulation starts by the procedure “start”



```
to start
;everything starts here - by button press
wait 100
initialize
while [1=1] [
make "t :t + 0.5
print se "t= :t
wait ask "delay [slider.value]
instance
]
end
```

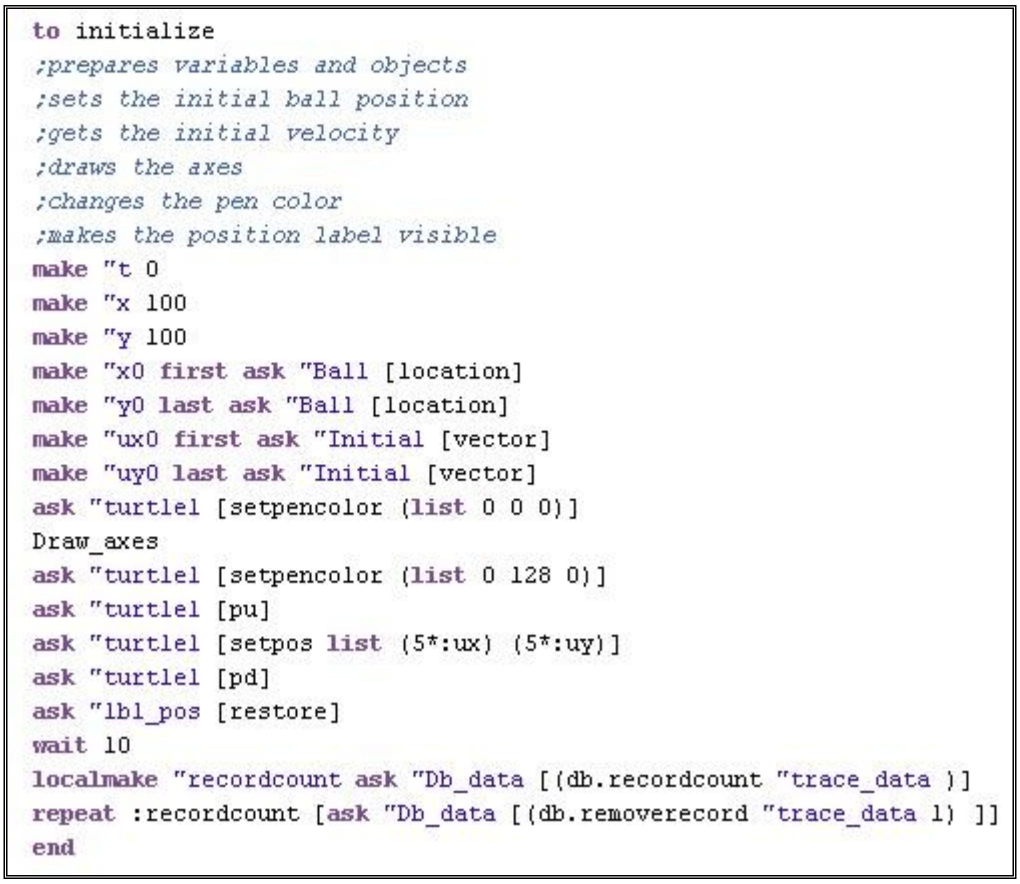
The procedures define_mass and define_radius update the scene object about the properties of respective sliders. These procedures are activated when the value of a slider is changing.



```
to define_mass ;updates the mass fo every slider change
localmake "mass_value ask "mass [slider.value]
ask "Ball [setmass :mass_value]
end

to define_radius ;;updates the radius fo every slider change
localmake "radius_value ask "radius [slider.value]
ask "Ball [setradius :radius_value]
end
```

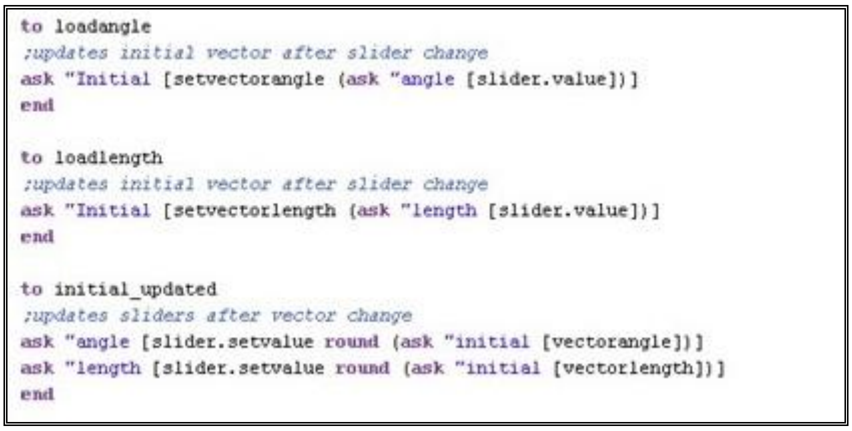
.

In the procedure «initialize» the initial values of motion and velocity are set. Furthermore, the graph axes are drawn (for future use) and data from previous simulation is being deleted.



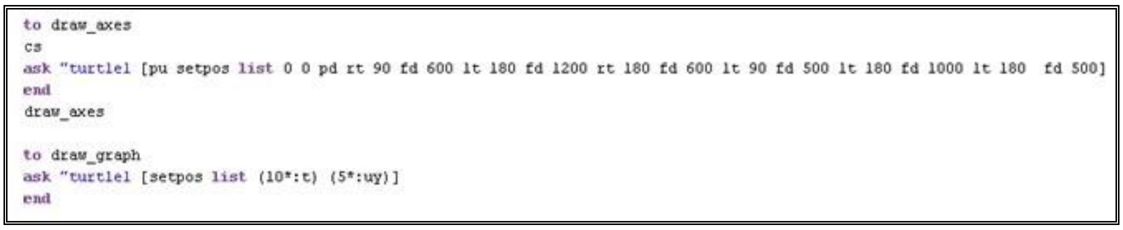
```
to initialize
;prepares variables and objects
;sets the initial ball position
;gets the initial velocity
;draws the axes
;changes the pen color
;makes the position label visible
make "t 0
make "x 100
make "y 100
make "x0 first ask "Ball [location]
make "y0 last ask "Ball [location]
make "ux0 first ask "Initial [vector]
make "uy0 last ask "Initial [vector]
ask "turtle1 [setpencolor (list 0 0 0)]
Draw_axes
ask "turtle1 [setpencolor (list 0 128 0)]
ask "turtle1 [pu]
ask "turtle1 [setpos list (5*:ux) (5*:uy)]
ask "turtle1 [pd]
ask "lbl_pos [restore]
wait 10
localmake "recordcount ask "Db_data [(db.recordcount "trace_data )]
repeat :recordcount [ask "Db_data [(db.removerecord "trace_data 1) ]]
end
```

The following procedures inform the sliders for any changes to vector and vice versa (vector with new value from the sliders) and are activated by respectively "events" of the components providing a two-way communication between the slider and the vector.



```
to loadangle
;updates initial vector after slider change
ask "Initial [setvectorangle (ask "angle [slider.value])]
end

to loadlength
;updates initial vector after slider change
ask "Initial [setvectorlength (ask "length [slider.value])]
end

to initial_updated
;updates sliders after vector change
ask "angle [slider.setvalue round (ask "initial [vectorangle])]
ask "length [slider.setvalue round (ask "initial [vectorlength])]
end
```

The procedures «draw_axes» and «draw_graph» form the axes and draw the graph.



```
to draw_axes
cs
ask "turtle1 [pu setpos list 0 0 pd rt 90 fd 600 lt 180 fd 1200 rt 180 fd 600 lt 90 fd 500 lt 180 fd 1000 lt 180  fd 500]
end
draw_axes

to draw_graph
ask "turtle1 [setpos list (10*:t) (5*:uy)]
end
```