ACTA DIDACTICA NAPOCENSIA
Volume 6, Number 1, 2013

A PLATFORM FOR DEVELOPMENT OF MATHEMATICAL GAMES ON SILVERLIGHT

Davorka Radaković and Đorđe Herceg

Abstract: Dynamic geometry software (DGS) is often used for development of interactive teaching materials in many subjects, not only mathematics. These interactive materials can contain hundreds of elements in order to represent complex objects, and script programs to control their behavior. We propose an approach for creating, importing and using components in the SLGeometry DGS, which can help solve the problem of overcomplicated geometric drawings. This approach benefits the teachers who can use ready-made components to develop their materials. We present an overview of our components, and a demonstration of their use. An experiment was conducted with a group of mathematics students in order to test our software in practice. The results of the experiment are also presented.

Key words: Geometry software, controllers, components, GeoGebra, SLGeometry

1. Introduction

GeoGebra [1] was conceived as a platform for development of interactive, dynamic geometric constructions, and it fulfills this role excellently. It is free, available in many languages, on all major operating systems, and used by numerous teachers from all around the world. It has been used to develop teaching materials not only for mathematics and geometry, but also for other subjects. GeoGebra has been rapidly evolving and gaining new features over the years, but its basic principle of operation is simple: shapes on screen are represented by expressions, which can depend on other expressions. The result is interactivity – when, for example, the user moves one point, everything connected to that point also moves. Many authors use GeoGebra for development of mathematical games, and games which are mathematical in nature. However, development of such games sometimes requires complex constructions and programming skills. This can pose a problem to school teachers, who usually do not possess the necessary skills.

We have been using GeoGebra to teach mathematics, especially numerical mathematics, and geography, and have developed a number of interactive teaching materials in it [2-5]. During our years with GeoGebra, we became aware of its many good sides, but also encountered some of its shortcomings, which are discussed in this paper. This motivated us to try and find ways to overcome them. We started developing SLGeometry, an extensible dynamic geometry framework, which runs on Microsoft Silverlight, [6-9]. Our aim is to use SLGeometry as a foundation for experiments in improving the development of teaching materials and games. Since SLGeometry is under our control, we are able to modify it, introduce new features, and develop some new principles of operation.

SLGeometry can import and use software components from DLL files. These components are either *interactive visual controls* (UI controls) or *sequential behavior controllers*. UI controls represent objects such as buttons, light bulbs, clocks, geographic maps etc. Behavior controllers contain control logic, which is employed to control the behavior of interactive drawings. We implemented a set of components, which facilitate development of interactive materials, especially mathematical games.

In this paper we present a pattern for building mathematical games in SLGeometry, and describe the features of our components, which help teachers develop games without programming. We conducted an experiment with a group of students of mathematics, in order to test how our approach compares to the existing practice.

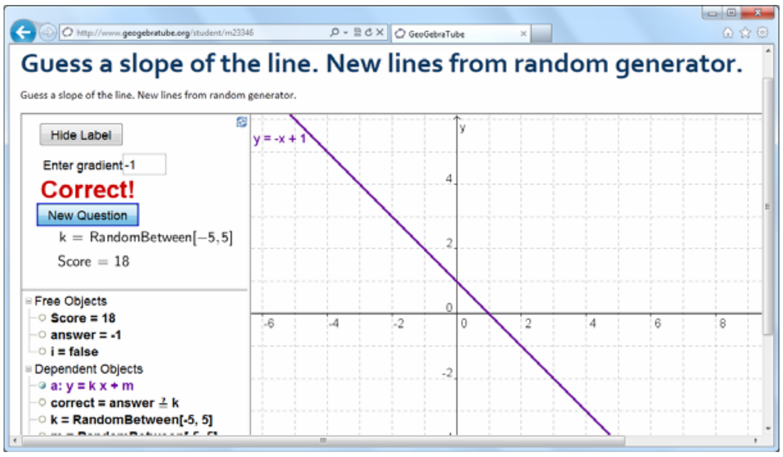
Received 28 February 2013, published 31 May 2013.

This paper is organized as follows: Section 2 describes a pattern of behavior which mathematical games, created in GeoGebra, often follow. An example is provided and explained. Section 3 deals with representation of complex visual objects in GeoGebra and provides examples of the problems which are usually encountered during development. Together, sections 2 and 3 explain the motivation behind our work. Section 4 presents interactive UI controls which we developed for SLGeometry. Section 5 presents sequential behavior controllers, which make the development of games without programming possible. This is also the main contribution. Section 6 contains the results of the experiment. Section 7 finishes with a conclusion and an outlook to future work.

2. Mathematical games in teaching and GeoGebra

The use of mathematical games in the teaching mathematics has been extensively discussed and researched. It is generally accepted that the use of games in class is justified, as it can motivate and encourage students to experiment and learn. There exist many titles on the use of games, multimedia and DGS in the teaching and learning of mathematics [10-19]. GeoGebra is used by numerous teachers around the world, who publish their teaching materials on GeoGebraTube [20]. We focus our discussion on GeoGebra and SLGeometry.

By searching the GeoGebraTube for terms like "game" and "guess", we can find many games, such as "Guess a slope of the line" [21], shown in Figure 1. The goal of this game is to guess the slope of a line, which is generated randomly in the slope-intercept form. The user has to enter the slope into a text box and click a button. The computer checks the result and, if it is correct, awards points to the user. The process is then repeated. Many other games, found on GeoGebraTube, follow the same pattern.


Guess a slope of the line. New lines from random generator.

Guess a slope of the line. New lines from random generator.

Enter gradient -1

Correct!

k = RandomBetween[−5, 5]

Score = 18

Free Objects
- Score = 18
- answer = -1
- i = false

Dependent Objects
- a: y = k x + m
- correct = answer ≟ k
- k = RandomBetween[-5, 5]

Figure 1. *A simple mathematical game in GeoGebra*

Many mathematical games can be developed by following a particular pattern, outlined here:

1. Describe the problem;
2. Devise the mathematical representation of the problem, and identify the values it depends on;
3. Designate parameters and allowed value ranges. Some parameters define the problem and are inaccessible to the player. The others form the solution, and can be modified by the player;
4. Draw the interactive construction, based on the parameters, in DGS;
5. Define the stopping criterion. This is usually a Boolean function, which yields False until the problem is solved. It then yields True;
6. Define the scoring function. This is usually a numeric function, which returns a value from the interval [0, 1], [0, 100] or similar;
7. Assign random values to the defining parameters, thus generating a concrete problem for the player to solve;
8. Let the player play the game by modifying free parameters until the stopping criterion is reached or an explicit "stop" command is given;
9. Display the score, and possibly add it to the total score.

The game can be repeated by repeating steps 7 to 9. Each time the problem will be different, because the defining parameters will be assigned different random values.

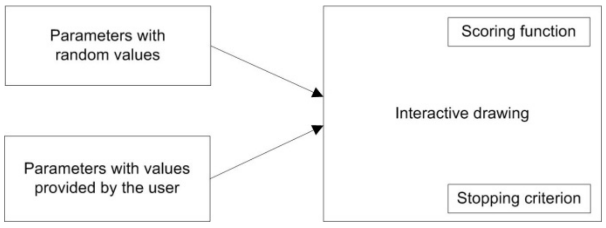

Parameters with random values
Parameters with values provided by the user
Scoring function
Interactive drawing
Stopping criterion

Figure 2. *A concept of a mathematical game in a DGS*

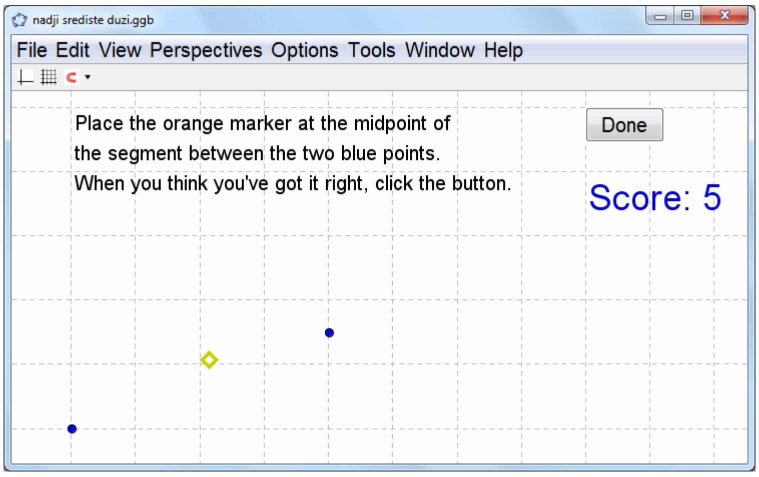
Due to the interactive nature of DGS, the stopping criterion and the scoring function are re-evaluated constantly, as the user changes the parameters. The parameters can be changed in a number of ways: by moving points, lines and other geometric shapes, by moving sliders or entering values into text boxes. The visibility of the score can be linked to the stopping criterion, i.e. the score is kept hidden until the criterion yields True. Usually, the message "You are correct" (or similar) then also appears.

A mathematical game in a DGS can be considered as a scoring function with input consisting of a set of randomly generated parameters and a set of parameters controlled by the user (Figure 2). While playing the game, the user aims to find appropriate parameter values in order to maximize the value of the scoring function.

However, once the correct solution is reached, the user can spoil it by simply continuing to change the parameters. In practice, our students often took advantage of this feature, and randomly moved objects in the drawing until they hit the right solution. From the students' point of view, this behavior is a very practical one, since a correct solution can be discovered without too much work.

In order to prevent such behavior, authors of mathematical games in GeoGebra resort to script programming. GeoGebra commands or JavaScript functions can be executed whenever a value of a variable changes or when an object is clicked with the mouse. This approach offers greater flexibility to the developer. With scripting, an interactive drawing can behave in a sequential manner, i.e. the behavior then depends not only on input values but also on its state, which is kept in variables.

This concept is demonstrated with a simple game, shown in Figure 3. The goal of the game is to guess the correct location of the midpoint of a segment between two points. Both the segment and its midpoint are invisible. One point is fixed, and the other point is placed randomly by the computer. The user must place the diamond marker within a given tolerance of the imagined midpoint. The scoring function, which depends on the distance between the marker and the real midpoint, returns 1 if the user has placed the marker close enough to the invisible midpoint. Otherwise, it returns 0. The marker is denoted with *X*, and the distance is defined as `dist = Segment(X, Midpoint(Segment(A,B)))`.



Place the orange marker at the midpoint of the segment between the two blue points. When you think you've got it right, click the button.

Figure 3. *The "guess the midpoint" game*

However, we want the game to behave in a slightly more complex manner, so that several rounds can be played, with a different random problem each time. The overall score will be the sum of scores from each individual round. In this setup, we must have a way to carry the cumulative score over from one round to another. Also, the user must be able to signal that he/she has solved the current problem and wants to move to the next one. In order to implement this behavior, we keep the total score in a variable. Furthermore, we provide the "Done" button, which the user has to click when he/she is ready to advance to the next problem. At each click, a script program updates the score and generates a new problem by moving the random point to a new location (Listing 1).

Listing 1. *The script that runs when the "Done" button is clicked*

`Score = Score + If(dist<1, 1, 0)`	Current value of the scoring function is added to the total score;
`B = (RandomBetween(1,8),RandomBetween(1,6))`	The position of the random point is changed.

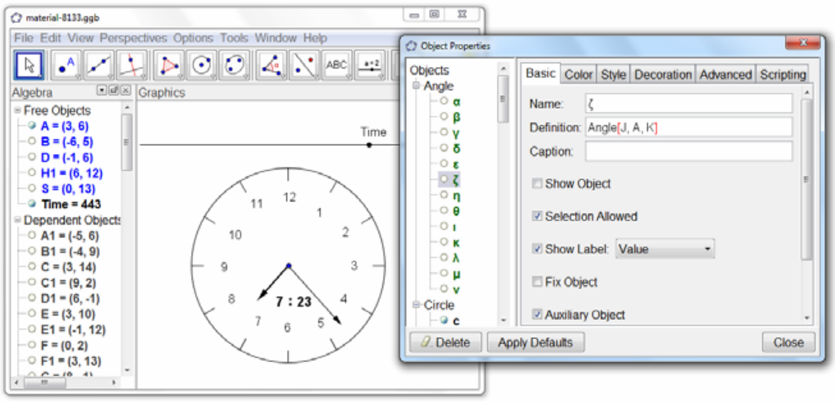
A stopping criterion is added in a similar way, by implementing a simple counter which is increased with each problem. When the counter reaches a predefined limit, the game is declared complete, and the "Done" button is hidden.

It should be noted here that this example is intentionally very simple. Our goal is to demonstrate the concepts and patterns involved in creating mathematical games, therefore we focused on the behavior, rather than geometric or mathematical complexity of the drawing.

The difficulty with this approach stems from the fact that the developer needs to be familiar with script programming. Furthermore, scripts can be assigned to many objects in the drawing, making following the control logic problematic. This limits the use of scripting to experienced developers and programmers. Unfortunately, school teachers are usually not skilled enough to develop games with such behavior.

3. Drawing complex objects in GeoGebra

There are many interactive materials on GeoGebraTube, which are not strictly mathematical in nature, or fall under other subjects, such as geography. The examples include a simple clock [22], and a 3D map of the world [23]. These drawings are created from many basic geometric shapes, which contribute to the visual appearance, but also burden the construction, because each shape must be assigned to a separate variable in GeoGebra. The clock (Figure 4), although simple in appearance, consists of 88 objects.

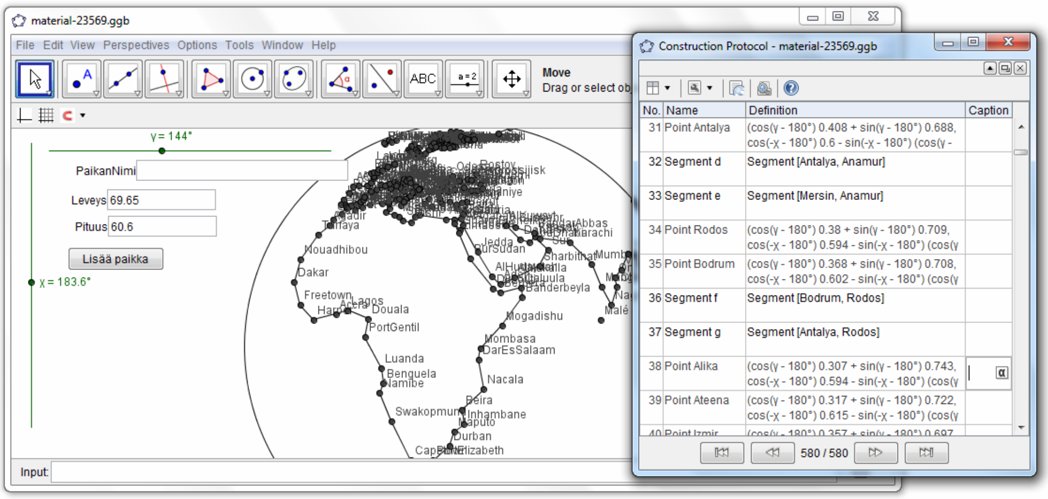

material-8133.ggb
File Edit View Perspectives Options Tools Window Help
Algebra
Graphics
Time
7 : 23
Object Properties
Objects
Angle
α β γ δ ε ζ η θ ι κ λ μ ν
Circle
c
Basic Color Style Decoration Advanced Scripting
Delete
Apply Defaults
Close

Free Objects
A = (3, 6)
B = (-6, 5)
D = (-1, 6)
H1 = (6, 12)
S = (0, 13)
Time = 443
Dependent Objects
A1 = (-5, 6)
B1 = (-4, 9)
C = (3, 14)
C1 = (9, 2)
D1 = (6, -1)
E = (3, 10)
E1 = (-1, 12)
F = (0, 2)
F1 = (3, 13)

Name: ζ
Definition: Angle[J, A, K]
Caption:
☐ Show Object
☑ Selection Allowed
☑ Show Label: Value
☐ Fix Object
☑ Auxiliary Object

Figure 4. *The drawing of a clock, which consists of 88 objects*

The 3D map of the world is an extreme example (Figure 5), because it contains as many as 580 objects, and yet it looks very rudimentary. A more detailed map would consist of thousands of points and line segments.



No.	Name	Definition	Caption
31	Point Antalya	(cos(γ - 180°) 0.408 + sin(γ - 180°) 0.688, cos(-χ - 180°) 0.6 - sin(-χ - 180°) (cos(γ -	
32	Segment d	Segment [Antalya, Anamur]	
33	Segment e	Segment [Mersin, Anamur]	
34	Point Rodos	(cos(γ - 180°) 0.38 + sin(γ - 180°) 0.709, cos(-χ - 180°) 0.594 - sin(-χ - 180°) (cos(γ	
35	Point Bodrum	(cos(γ - 180°) 0.368 + sin(γ - 180°) 0.708, cos(-χ - 180°) 0.602 - sin(-χ - 180°) (cos(γ	
36	Segment f	Segment [Bodrum, Rodos]	
37	Segment g	Segment [Antalya, Rodos]	
38	Point Alika	(cos(γ - 180°) 0.307 + sin(γ - 180°) 0.743, cos(-χ - 180°) 0.594 - sin(-χ - 180°) (cos(γ	
39	Point Ateena	(cos(γ - 180°) 0.317 + sin(γ - 180°) 0.722, cos(-χ - 180°) 0.615 - sin(-χ - 180°) (cos(γ	

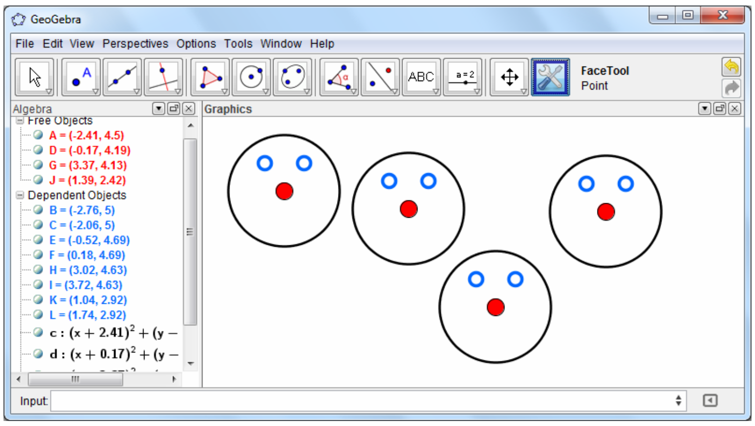
Figure 5. *The 3D map of the world consists of 580 objects*

It is easy to see that development of complex interactive drawings can be difficult, if they represent something other than geometry. We have identified two main problems. Firstly, there is no clear logical connection between a complex object shown on the screen and the many shapes that make it up. Secondly, duplicating such an object effectively doubles the number of variables. This problem escalates if several copies of a complex object are needed in a drawing.

It quickly becomes very difficult for the author to keep track of all the variables while developing the interactive drawing. More specifically, it is not easy to distinguish between important and unimportant shapes. The clock in Figure 4 can be taken as an example. The angle between the clock hands is important, but the angles which were used to place numbers on the clock face are not. Yet, when we look at the list of variables, all the angles look similar.

GeoGebra addresses the problem of logical grouping with *user defined tools*. A tool acts as a group of several shapes, and it is created by specifying input and output objects from the drawing. Once created, a tool can be used with the mouse and as a text command. However, the second problem remains, as each instance of the tool still creates a number of variables.

We can demonstrate this behavior with a simple example. In Figure 6, we created a stylized face from three points and a circle, and grouped them into a tool, called "FaceTool". Then we created four instances of the tool, each allocating three variables for the points and one variable for the circle. This results in definitions of twelve points and four circles. In the Algebra view, it is not clear that these twelve variables represent four faces, or which face they belong to.



Free Objects
- A = (-2.41, 4.5)
- D = (-0.17, 4.19)
- G = (3.37, 4.13)
- J = (1.39, 2.42)

Dependent Objects
- B = (-2.76, 5)
- C = (-2.06, 5)
- E = (-0.52, 4.69)
- F = (0.18, 4.69)
- H = (3.02, 4.63)
- I = (3.72, 4.63)
- K = (1.04, 2.92)
- L = (1.74, 2.92)
- c : $(x + 2.41)^2 + (y -$
- d : $(x + 0.17)^2 + (y -$

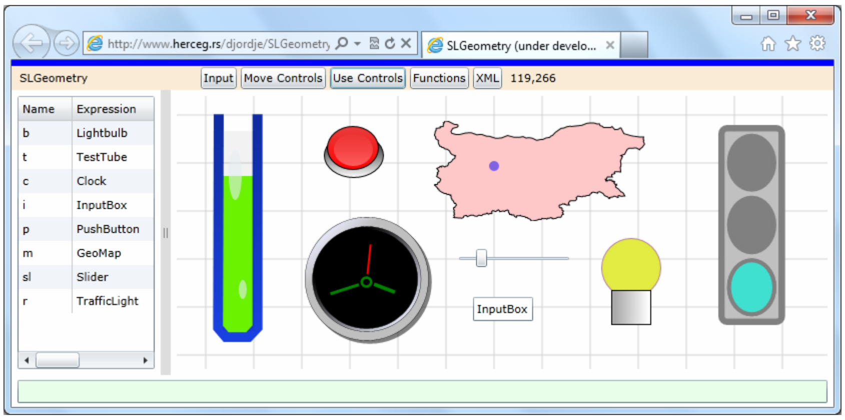
Figure 6. *Multiple instances of the FaceTool produce many constituent objects in GeoGebra*

We concluded that a simpler way of handling complex objects would benefit the authors, as it would enable them to concentrate more on the general idea, instead of keeping track of tiny details of their drawings. The first step we took was to enable the use of Silverlight UI controls in SLGeometry.

4. Interactive UI controls in SLGeometry

A Silverlight UI control encapsulates visual appearance, interactivity and program code in one object. In order to develop a UI control, a development environment, such as Microsoft Visual Studio or Microsoft Expression Studio, is needed. The developer must have a basic knowledge of C# programming and Silverlight graphics design. However, control development is not particularly difficult, and there are many examples on the web, such as [24]. More detail about how UI controls are developed, packaged into library files, imported into SLGeometry and used in geometric drawings is provided in [9].

A number of standard UI controls already exist in Silverlight, such as buttons, text boxes, sliders and check boxes. We developed several custom UI controls, representing the objects which are often used in teaching materials and games found on GeoGebraTube. They include a light bulb, a test tube, a button, a clock, a traffic light and a geographic map (Figure 7). Each UI control is represented by a single variable in SLGeometry, and has a number of properties, which control its appearance and behavior. Some UI controls react to user input, either by mouse or keyboard, and the changes are propagated to appropriate properties.

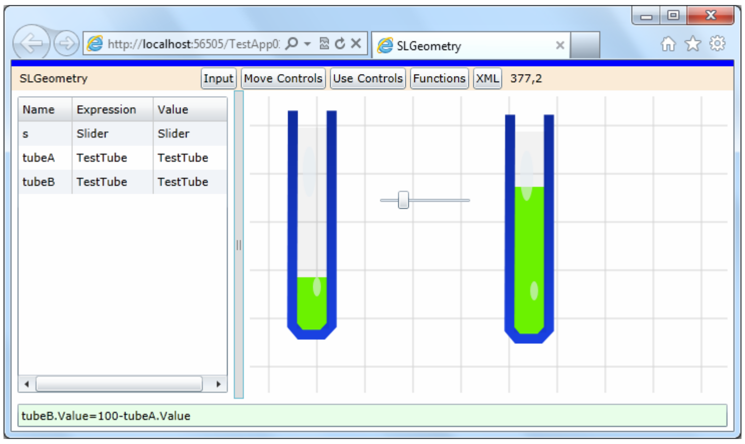


Name	Expression
b	Lightbulb
t	TestTube
c	Clock
i	InputBox
p	PushButton
m	GeoMap
sl	Slider
r	TrafficLight

Figure 7. *Visual components are single objects in SLGeometry*

Custom UI controls, which we developed for SLGeometry, are listed in Table 1, together with descriptions of their important properties. Most properties can be assigned a value or an expression, which in turn causes a change in the visual appearance of a control. For example, assigning *True* to the *Lit* property of a LightBulb causes the light bulb to appear lit. Properties marked as *read only* can not be assigned to.

Properties marked as *two way* can also be changed by using the keyboard and the mouse. For example, the *Value* property of a slider changes when the user drags the handle with the mouse.



Name	Expression	Value
s	Slider	Slider
tubeA	TestTube	TestTube
tubeB	TestTube	TestTube

```
tubeB.Value=100-tubeA.Value
```

Figure 8. *Both test tubes depend on the value of the slider*

One important feature of SLGeometry is that expressions, which refer to properties of UI controls, can be assigned to properties of other UI controls. This makes propagation of values from one control to another possible.

Let us consider a simple example in Figure 8. It contains a slider, named *s*, and two test tubes - *tubeA* on the left and *tubeB* on the right. The range of the slider is set to [0, 100], by assigning values to its *Minimum* and *Maximum* properties. With the command `tubeA.Value=s.Value` we connect the "output" of the slider with the "input" of the first test tube, which enables us to control the level of the liquid in it with the slider. Then, with the command `tubeB.Value=100-tubeA.Value`, we connect the "input" of the second test tube with the expression, which depends on the value of the first test tube. As we move the slider with the mouse, the level of liquids in both test tubes changes. Many variations like these are possible, making creating highly interactive drawings easy.

Table 1. *Custom UI controls and their properties*

Control/property	Type	Description
LightBulb		*A lightbulb which can be either on or off*
Lit	Logical	Controls whether the lightbulb appears lit or not
InputBox		*A box to type text into*
Text	String, two way	Text which appears in the InputBox
TestTube		*A test tube with an arbitrary level of liquid inside it*
Color	RGBColor	Sets the color of the liquid in the test tube
Value	Number, two way	Gets or sets the level of the liquid in the test tube
Clock		*An analog clock that can be set to show arbitrary time*
Hour	Number	Hours
Minute	Number	Minutes
Second	Number	Seconds
PushButton		A button that can be pressed with the mouse
Pressed	Logical, read only	Indicates whether the mouse button is pressed over the button
GeoMap		*Shows country maps in cylindrical projection*
Country	String	Sets the country which is to be displayed on the map
MapColor	RGBColor	Sets the color of the map
Marker	Point, two way	Gets or sets the location of the marker on the map
MarkerSize	Number	Sets the size of the marker
Normal	Logical	Controls whether the map can stretch to fill the entire control
Width	Number	Gets or sets the width of the map in pixels
Height	Number	Gets or sets the height of the map in pixels
Points	List	Sets the additional markers to be placed on the map
Slider		*A numerical value can be set with the mouse*
Minimum	Number	Gets or sets the minimum value of the slider
Maximum	Number	Gets or sets the maximum value of the slider
Width	Number	Gets or sets the width of the slider in pixels
Value	Number, two way	Gets or sets the current value of the slider
TrafficLight		*A simple traffic light with three lights*
Light	Number	Controls the lights: 0 – none, 1 – red, 2 – yellow, 3 - green
IsRed	Logical, read only	Indicates whether the red light is lit
IsYellow	Logical, read only	Indicates whether the yellow light is lit
IsGreen	Logical, read only	Indicates whether the green light is lit

5. Sequential behavior controllers in SLGeometry

Further extending our previous work with custom components, we developed behavior controller components for SLGeometry, which act in a sequential manner, i.e. they have inputs, memory and outputs. With controllers, it is possible to achieve the most common behavior needed in mathematical games, without the need for programming. Instead, the user places controllers on the drawing and sets their properties accordingly. Each type of controller performs a specific type of action, based on input values, and produces the appropriate output. The action does not occur immediately, however. Instead, it is triggered by assigning the logical value *True* to the special *Trigger* property of a controller. Controller's outputs are updated immediately afterwards. A special output property, named *Done* is then briefly set to True and then again to False. The controllers can be daisychained together, by connecting the *Done* output of one controller to the *Trigger* input of another. If used correctly, this enables users to create sequential behavior without the need for writing program code.

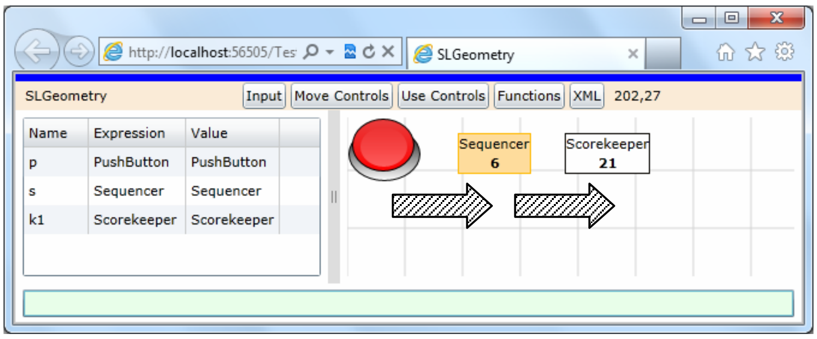
Table 2. *Sequential behavior controllers and their properties*

Controller/property	Type	Description
Sequencer		*A counter which cycles through an interval of integer values.*
Min	Number	Lower bound of the interval
Max	Number	Upper bound of the interval
Value	Number	Current value of the Sequencer
Trigger	Logical	On transition from False to True, the trigger causes *Value* to increase by 1. After *Max* is reached, the Sequencer starts again from *Min*.
Done	Logical, read only	Transitions from False to True to False after Value is updated
Randomizer		*Produces a random value from the specified interval*
Min	Number	Lower bound of the interval
Max	Number	Upper bound of the interval
Trigger	Logical	On transition from False to True, causes the Randomizer to generate a new *Value*
Done	Logical, read only	Transitions from False to True to False after Value is updated
Scorekeeper		*Adds points to the total score when triggered*
Points	Number	Score in the current round of a game
Score	Number, read only	Total score
Trigger	Logical	On transition from False to True, the value of *Points* is added to *Score*.
Done	Logical, read only	Transitions from False to True to False after Value is updated

The following examples demonstrate the practical use of the controllers.

Example 1 – Arithmetic sum

The following example (Figure 9) demonstrates the use of triggers and their propagation from one component to another. The goal of this example is to calculate the arithmetic sum of numbers from 1 to 9. We use the Sequencer to produce values from 1 to 9, and the Scorekeeper to add them together. A Pushbutton triggers the Sequencer, and the Sequencer's *Done* property then triggers the Scorekeeper. The current value of the Sequencer is fed into the *Points* property of the Scorekeeper.



Name	Expression	Value
p	PushButton	PushButton
s	Sequencer	Sequencer
k1	Scorekeeper	Scorekeeper

Figure 9. *Propagation of triggers between components*

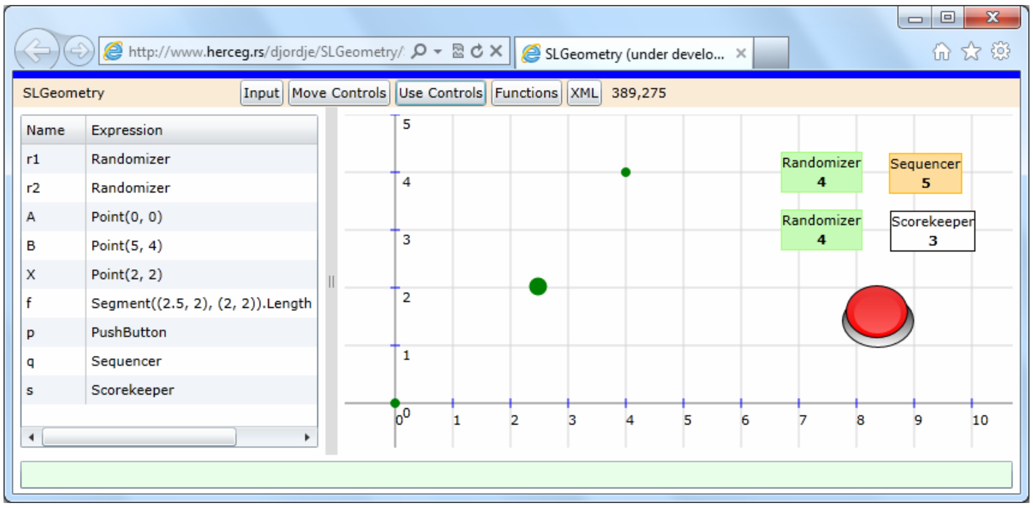
Clicking the button causes the Sequencer to cycle through numbers from 1 to 9, which are then added to the score, thereby producing an arithmetic sum as a result. In order to achieve this behavior, the user needs only to connect the controllers to each other, as shown in Listing 2.

Listing 2. *Daisychaining the components via triggers*

`s.Trigger = p.Pressed`	The Sequencer's trigger is connected to the button, so that it advances each time the button is clicked.
`k1.Points = s.Value`	The current value of the Sequencer is connected to the input of the Scorekeeper.
`k1.Trigger = s.Done`	The Scorekeeper adds *Points* to the sum right after the Sequencer finishes.

Example 2 – A simple game

This example demonstrates (Figure 10) the implementation of the "guess the midpoint" game in SLGeometry. The game is set up in a similar way as in GeoGebra, with one important difference – the behavior is controlled without script programs. Instead, the game logic is implemented by using controllers. Randomizers *r1* and *r2* provide random coordinates for the point *B*. Sequencer *q* keeps track of how many times the game has been played. PushButton *p* signals that the user has placed the marker and wants to check his/her solution. When the button is clicked, Scorekeeper *s* increases by 1 if the user's solution is correct. Scorekeeper's *Done* property then triggers the Sequencer and the Randomizers, thus moving the point B to a new random location and starting a new round of the game.



Name	Expression
r1	Randomizer
r2	Randomizer
A	Point(0, 0)
B	Point(5, 4)
X	Point(2, 2)
f	Segment((2.5, 2), (2, 2)).Length
p	PushButton
q	Sequencer
s	Scorekeeper

Figure 10. *The "guess the midpoint" game in SLGeometry*

The complete code that produces the game is shown in Listing 3. It should be noted that the code is provided only for the sake of completeness, as the users will create the game using toolbars and mouse. Important commands, which provide sequential behavior, are shown in boldface.

Listing 3. *Definition of the "guess the midpoint" game*

`r1 = Randomizer()`
`r2 = Randomizer()`
`r1.Min = 2`
`r1.Max = 8`
`r2.Min = 1`
`r2.Max = 7`
	Two Randomizers are created and their intervals set.

`A = (0,0)`
`B = (r1.Value, r2.Value)`
`X = (2,2)`
`X.Size = 15`
	Point A is fixed.
	Point B is placed randomly.
	Point X is supposed to be moved by the user, so we make it appear bigger.

`f = Segment(Segment(A,B).Midpoint, X).Length`	*f* is the distance between the point *X* and the midpoint of the segment *AB*.

`p = PushButton()`
`q = Sequencer()`
`s = Scorekeeper()`
	A PushButton, a Sequencer and a Scorekeeper are created.

`s.Points = If(f<1, 1, 0)`	Current score is fed into the Scorekeeper.

`s.Trigger = p.Pressed`
`q.Trigger = s.Done`
`r1.Trigger = s.Done`
`r2.Trigger = s.Done`
	Sequential behavior is defined here. The PushButton activates the Scorekeeper, which adds the current score. When the Scorekeeper finishes, it activates the sequencer and both randomizers, preparing the game for the next round.

6. Experiment

In order to test our behavior controllers in practice, we conducted an experiment with a group of 23 first year students of mathematics. We looked for answers to the following questions:

- Are behavior controllers easier to learn and use than script programming?
- Given the choice, how many participants would choose behavior controllers over script programs to implement a mathematical game?
- How well can the students solve a problem, which was not previously demonstrated, using the behavior controllers?
- Do the students find the behavior controllers significantly easier to use than scripting?

Assignments

The experiment was conducted during two classes, in a computer lab. In the first class, we taught the students script programming and behavior controllers, and presented the two examples from Section 5. Then we asked them to reproduce the examples, with our help. One week later, in the second class, the students were given two assignments:

1. Implement the "Guess the midpoint" game from Example 2, with two random points *A* and *B*:
    a. using script programming,
    b. using behavior controllers.
2. Implement the three-digit counter, which consists of three Sequencers. The counter should advance when a PushButton is pressed. (*A car mileage meter was shown as an example*)

The second assignment, which was not presented before, was to implement a multi-digit counter using the Sequencer control. Both assignments were graded from 1 (fail) to 4 (excellent). Finally the students were polled on their opinions about the behavior controllers.

The students were graded in the following manner:

- First, the students which had solved their assignments correctly, were graded 'Excellent'.
- The students which had made minor syntax errors were shown how to correct them. Such students were graded 'Good'.
- Finally, we gave additional instructions to the rest of the students. Then, the ones which successfully solved the problems were graded 'Satisfactory', and the rest received 'Fail'.

Table 3. Students' test scores

	Assignments		
	A1a	A1b	A2
Fail	6	2	0
Satisfactory	2	4	3
Good	8	7	6
Excellent	7	10	14

Test results are shown in Table 3. Columns "A1a" and "A1b" contain results from the first assignment, solved using scripting and behavior controllers respectively. Column "A2" contains results from the second assignment.

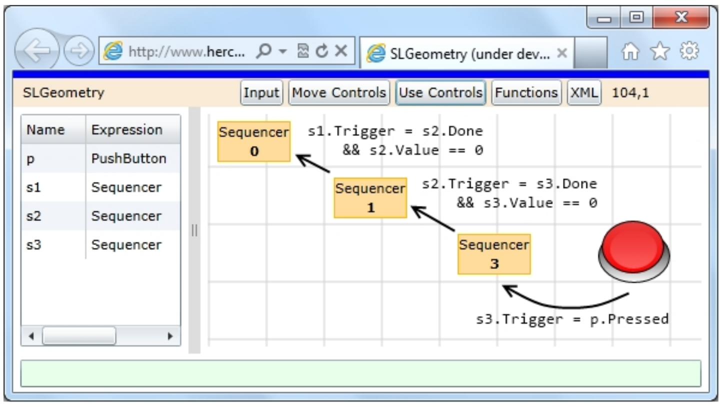
Regarding the first assignment, success rate was better with behavior controllers than with scripts. While eight students struggled with script programming and subsequently six of them failed to complete the assignment, only two failed to complete the same assignment using controllers. The number of excellent results was also bigger with the controllers (10 versus 7).

The amount of help we had to provide was also different in favor of controllers. In A1a, we had to explain both the syntax and semantics of the script commands, while in A1b we only had to remind the students how to correctly address controller properties.

The second assignment was succesfully completed by all the students. As with the previous assignment, we helped the students, which scored 'Good', by correcting minor syntax errors. The three students which scored 'Satisfactory' could not formulate the correct criterion for triggering the next digit by themselves, so we helped them with the explanation:

When one Sequencer transitions from 9 to 0, the next Sequencer should increase by 1.

The students then easily translated this to a logical condition which triggers the next digit, and solved the problem (Figure 11).



Name	Expression
p	PushButton
s1	Sequencer
s2	Sequencer
s3	Sequencer

```
s1.Trigger = s2.Done
  && s2.Value == 0
s2.Trigger = s3.Done
  && s3.Value == 0
s3.Trigger = p.Pressed
```

Figure 11. *Propagation of triggers in the second assignment*

Poll results

We polled the students on their opinions about our behavior controllers. The questions and answers are presented in Table 4.

Table 4. Poll results

Question	Yes	No
Are behavior controllers easier to use than script programs?	23	0
Do you find one class enough to learn behavior controllers?	20	3
Do you find one class enough to learn script programming?	6	17
If you had to choose only one method of controlling the behavior of your interactive drawing, would you choose behavior controllers over scripting? Explain.	15	8
Your opinion on behavior controllers, in writing?	-	-

We conclude from the answers that all the participants recognized the simplicity that the behavior controllers offer. Most of the students learned how to use them without difficulty. The opposite holds for script programming. Learning how to program was difficult for students, especially for those who had no previous programming knowledge. However, approximately one third of the students recognized that scripting offers more choice and versatility. They stated that they were confident enough in their programming skills to use script programming instead of controllers.

The participants answered the last question in writing, and later we had a discussion with them, which resulted in some interesting opinions and observations:

- "I can actually *see* what the controllers do. Scripts are invisible and not very clear to me."
- "With controllers, I don't have to memorize commands."
- "Scripting is more powerful, but too difficult for me."
- "I wish you created more behavior controllers for more complex games."

While most students agreed that the controllers are a useful addition to a DGS, some argued that they did not offer enough flexibility. On the other hand, more than half of the students said that they were able to create a simple mathematical game with controllers, while they could not do the same with scripting. As the students of mathematics are the future teachers of mathematics, we find that this conclusion justifies the use of behavior controllers.

7. Conclusions and further work

Many teachers are developing interactive teaching materials and mathematical games using dynamic geometry software. During development, they often face difficulties which can be overcome by implementing new features into the software. We propose an approach based on interactive components in the SLGeometry DGS, which helps the teachers develop their materials more easily. Our components simplify the development and control of the behavior of complex drawings.

Development of custom visual controls requires certain knowledge of programming. However, advanced users who are proficient in writing scripts in GeoGebra, should be able to start creating controls with minimal effort. The others will be able to download compiled control libraries and use them in their drawings.

We identified a pattern of behavior, common to many mathematical games created in GeoGebra. The behavior of such games is usually controlled by script programs, which can be difficult to write and maintain. We developed several sequential behavior controllers, which help in creation of mathematical games in SLGeometry DGS without the need for programming. It is our aim to develop additional controllers, to enable the users to create games with more complex behavior. The use of controllers needs not be limited to games only. Since SLGeometry is an open and extendable platform, anyone with enough programming skills can develop their own components. Again, anyone can download and use ready-made controllers in their drawings.

A small-scale test and a poll were conducted with a group of students of mathematics. The controllers were received with enthusiasm and interest by the students. They were able to solve the assignments more succesfully using our behavior controllers, than with scripting. They also felt that the controllers were not as intimidating as programming. The test and the poll confirmed our view that behavior controllers can be applied in practice and that they have a place in DGS. The main result of the experiment shows that the behavior controllers are easy to use, even by those individuals, who do not possess programming skills. We plan to conduct further tests on a larger scale.

SLGeometry is still in development. Currently we are developing features that will enable the users to utilize SLGeometry in class and for development of teaching materials.

The authors would like to thank the unknown referees for their constructive comments.

References

[1] GeoGebra - http://www.geogebra.org, [September 1, 2012]

[2] Herceg, D., Herceg, Đ. (2008): Numerical Mathematics with GeoGebra in High School, *Teaching Mathematics and Computer Science 6/2*, pp. 363-378

[3] Herceg, D., Herceg, Đ. (2009): Interpolation with GeoGebra, *Pedagoška stvarnost*, LV, 9-10, pp. 897-908

[4] Herceg, Đ., Herceg, D. (2010): Numerical Integration with GeoGebra in High School, *The International Journal for Technology in Mathematics Education* – 17(4), pp. 205-210

[5] Herceg-Mandić, V., Herceg, Đ. (2011): Spatial orientation with GeoGebra, *Proceedings of the International GeoGebra Conference for Southeast Europe*, Đ. Takači, ed., Prirodno-matematički fakultet u Novom Sadu, Novi Sad, pp. 39-52

[6] Radaković D., Herceg, Đ., Löberbauer, M. (2010): Extensible expression evaluator for the dynamic geometry software Geometrijica, *PRIM 2009*, Novi Sad, pp. 95-100

[7] Radaković, D., Herceg, Đ. (2010): The Use of WPF for Development of Interactive Geometry Software, *Acta Univ. M. Belii ser Mathematics 16*, pp. 65-79

[8] Herceg, Đ., Radaković, D. (2011): The Extensibility of an Interpreted Language Using Plugin Libraries, Numerical Analysis and Applied Mathematics ICNAAM 2011, *AIP Conf. Proc. 1389*, pp. 837-840

[9] Herceg, Đ., Radaković, D., Herceg, D.L. (2012): Generalizing the Extensibility of a Dynamic Geometry Software, Numerical Analysis and Applied Mathematics ICNAAM 2012, AIP Conf. Proc. 1479, pp. 482-485

[10] Ahl, D. (1981). Computer games in mathematics education. Mathematics Teacher, 74(8), pp. 653-656.

[11] Amory, A., Naicker, K., Vincent, J. & Adams, C. (1999). The use of computer games as an educational tool: identification of appropriate game types and game elements. British Journal of Educational Technology, 30(4), pp. 311-321.

[12] Communicating about Mathematics Using Games, National Council of Teachers of Mathematics, http://illuminations.nctm.org/LessonDetail.aspx?ID=U123 [January 10, 2013]

[13] Dempsey, J. V., Haynes, L. L., Lucassen, B. A., & Casey, M. S. (2002). Forty simple computer games and what they could mean to educators. Simulation & Gaming, 33(2), pp. 157-168.

[14] McDonald, K. K. & Hannafin, R. D. (2003). Using web-based computer games to meet the demands of today's high-stakes testing: A mixed method inquiry. Journal of Research on Technology in Education, 35(4), pp. 459-472.

[15] Nusir, S., Alsmadi, I., Al-Kabi, M., Sharadgah, F. (2012). Studying the Impact of Using Multimedia Interactive Programs at Children Ability to Learn Basic Math Skills. Acta Didactica Napocensia 5(2), pp. 17-32.

[16] Prensky, M. (2001). Digital game-based learning. McGraw-Hill, New York.

[17] Stahl, G., Çakir, M. P., Weimar, S., Weusijana, B. K., Ou, J. X. (2010). Enhancing Mathematical Communication for Virtual Math Teams. Acta Didactica Napocensia 5(2), pp. 1-13.

[18] Randel, J., Morris, B., Wetzel, C., Whitehill, B. (1992): The Effectiveness of Games for Educational Purposes: A Review of Recent Research, *Simulation & Gaming*, 23(3), pp. 261–276.

[19] Vankúš, P., (2008): Games Based Learning in Teaching of Mathematics, *Acta Didactica Universitatis Comenianae, Mathematics* 8, pp. 103-120.

[20] GeoGebraTube, http://www.geogebratube.org/ [September 1, 2012]

[21] Guess a slope of the line, http://www.geogebratube.org/student/m23346 [December 20, 2012]

[22] Clock24hour, http://www.geogebratube.org/material/show/id/8133 [January 10, 2013]

[23] 3D map of the world, http://www.geogebratube.org/material/show/id/23569 [January 10, 2013]

[24] Walkthrough: Creating a Silverlight Clock by Using Expression Blend or Visual Studio, http://msdn.microsoft.com/en-us/library/bb404709(v=vs.95).aspx [September 1, 2012]

Authors

Davorka Radaković, Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Serbia, e-mail: davorkar@dmi.uns.ac.rs

Đorđe Herceg, Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Serbia, e-mail: herceg@dmi.uns.ac.rs