

USING GEOGEBRA'S RANDOMIZATION AND JAVASCRIPT TO CREATE INTERACTIVE RANDOMIZED STUDENT EXERCISES

Marc Renault

Shippensburg University, Shippensburg, PA, USA, msrenault@ship.edu

Abstract: *We give three examples of web-based GeoGebra applets where students are presented with a problem that they must solve. Once they solve the problem, they can reset the applet to produce a similar problem but with some parameters randomized. These applets make use of some randomization commands native to GeoGebra, and some JavaScript coding in the html page. The applets presented in this paper may be found at http://webspaceship.edu/msrenault/Interactive_GeoGebra_Applets.html*

Keywords: *JavaScript, randomization, student activities, interactive applets*

1. INTRODUCTION

Using GeoGebra, one can produce excellent demonstrations of mathematical ideas and wonderful illustrations of problems. In this paper we explore another use for GeoGebra: creating web-based *interactive* applets that students engage with in order to solve a problem. Once the problem has been solved, the applet can be reset, some parameters are randomized, and the student is presented with a similar but different problem. These applets are interactive in a way that demonstrations and illustrations are not.

We'll look at three interactive applets that make use of GeoGebra's randomization commands and some JavaScript coding. Briefly, here are the problems that the applets present:

1. The equation for a line is randomly generated and the student must position a line on the screen so that it is the line of the equation.
2. A point is randomly generated and displayed on a polar grid. The student must type the polar coordinates of the point into an input box.
3. Four points are displayed, and four coordinate-pairs are randomly generated. The student must drag the points to the correct coordinates.

2. THE FIRST APPLET: CREATE A LINE WITH A GIVEN EQUATION

In this applet the student is presented with the equation for a line, and he/she must drag a given line so that it matches the equation. When this is done correctly the word "Good!" appears. See Figure 1. There is a button below the graphics window that the student can click to get a new, randomly generated equation.

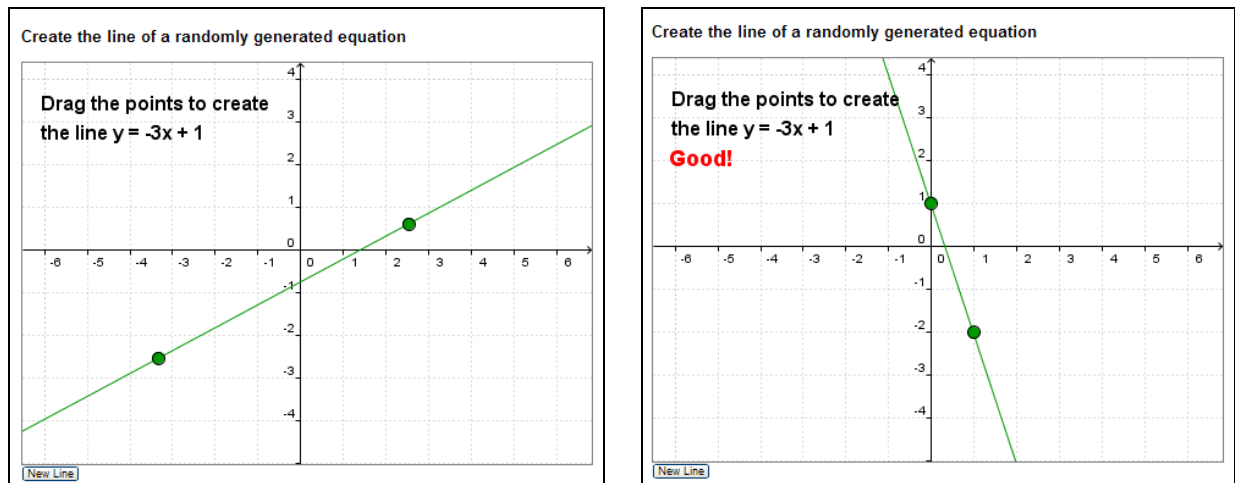


Figure 1: The first applet.

To create the GeoGebra file, points A and B were placed on the screen, a line was drawn between them, and then the following commands were entered, in the order shown.

```
slopeList = {-3, -2, -1, 0, 1, 2, 3, -2.5, -1.5, 1.5, 2.5, -1.66, -1.33, -0.66, -0.33, 0.33, 0.66, 1.33, 1.66}
```

```
m = Element[ slopeList, RandomBetween[1, Length[slopeList]] ]
```

```
b = RandomBetween[-3, 3]
```

```
hiddenLine: y = m*x + b
```

```
distA = Distance[A, hiddenLine]
```

```
distB = Distance[B, hiddenLine]
```

The line *hiddenLine* is, of course, hidden. Finally, text was created. The text “Good!” appears when the condition

$$distA < 0.08 \wedge distB < 0.08$$

holds true. The value of 0.08 was determined by trial and error, finding a value that would make the line on the screen a reasonable approximation to the hidden line.

When the objects are recomputed (either by selecting “Recompute All Objects” from the menu or by refreshing the applet in a web page), new randomization occurs and a new equation and hidden line are formed.

We’ll look at two methods for re-randomizing the variable values. The first method, resetting the applet, is the easiest but it has flaws. The second method, introducing a dummy variable, is an improvement.

1.1 Re-randomization by Resetting the Applet

After the GeoGebra file is exported to a web page, a button is added and JavaScript code is created to handle the button click. The following code is added to the webpage below the applet.

```
<form name="MyForm">
  <input type="button" value="New Line" onclick="makeNewLineEqn()" >
</form>
```

```
<script type="text/javascript">
  function makeNewLineEqn() {
    document.ggbApplet.reset();
  }
</script>
```

Clicking the button calls the function `makeNewLineEqn` which resets the applet, causing it to return to its initial state and recalculating all the random variables.

There are some problems with this method. The reset does not happen instantaneously and smoothly; it takes just under a second and there is often a distracting flicker as the applet reloads. Also, the line that the student has been dragging gets reset to its initial position instead of staying where it is. Ideally, the student should see *only* the equation change, and not be distracted by other changing objects. These problems can be solved by introducing a dummy variable.

1.2 Re-randomization by Introducing a Dummy Variable

In the GeoGebra file we define a new number, *dummy*, and modify two lines as shown.

```
dummy = 19
m = Element[ slopeList, RandomBetween[1, Length[slopeList]] + 0*dummy ]
b = RandomBetween[-3, 3] + 0*dummy
```

Then, in the JavaScript code we modify the function `makeNewLineEqn`.

```
function makeNewLineEqn() {
  document.ggbApplet.evalCommand("dummy = 19");
}
```

When the button is clicked, JavaScript tells the GeoGebra file to redefine the number *dummy*, and *every value that depends on dummy is recalculated*. The values for *m* and *b* depend on *dummy* (trivially), so GeoGebra recomputes those random values. Conceivably, one might have an applet with several dummy variables, and several buttons to re-randomize different parameters.

3. THE SECOND APPLLET: WRITE THE POLAR COORDINATES OF A GIVEN POINT

In the first applet, an algebraic representation of an object (a line) was given, and the student had to create a graphical equivalent. In this next applet the reverse takes place: the graphical representation is given (a point in the plane) and the student must come up with an algebraic equivalent. In this case the student types in the polar coordinates of the point. See Figure 2. There is a button to reset and re-randomize the problem, a text box for students to enter their answer, and a button to click to check the answer.

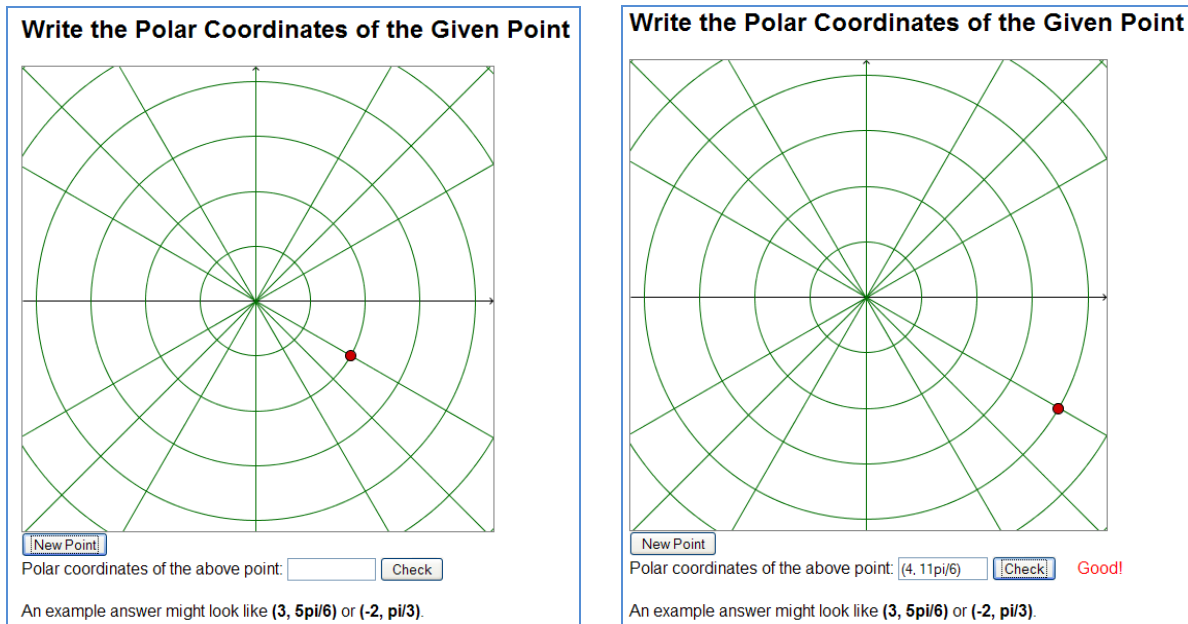


Figure 2: The second applet.

To create the GeoGebra file, the polar grid was drawn, then a point P was placed on the screen and hidden. The following commands create a random point Q with polar coordinates (r, θ) and determine whether P is “close enough” to Q .

```
dummy = 19
r = RandomBetween[1, 4] + 0*dummy
thetaList = {0, 1, 1/2, 3/2, 1/3, 2/3, 4/3, 5/3, 1/4, 3/4, 5/4, 7/4, 1/6, 5/6, 7/6, 11/6}*pi
thetaIndex = RandomBetween[1, Length[thetaList]] + 0*dummy
theta = Element[thetaList, thetaIndex]
Q = (r, theta)
tfCloseEnough = Distance[P, Q] < 0.1
```

Notice that GeoGebra multiplies π through the list of numbers in $thetaList$. Also, notice that GeoGebra uses a semicolon to denote points in polar coordinates. Finally, observe that $tfCloseEnough$ is a Boolean variable.

When the New Point button is clicked, JavaScript tells GeoGebra to redefine the dummy variable, and this causes point Q to be recomputed and displayed somewhere random on the screen. When the student clicks the Check button, JavaScript performs the following tasks:

- Take the student’s answer (s, t) and replace the comma with a semicolon to get $(s; t)$.
- Tell the GeoGebra file to execute the command $P = (s; t)$.
- Look at the value of $tfCloseEnough$ and output either “Good!” or “No”.

In this applet we use a `` html element to write dynamic text next to the Check button.

4. THE THIRD APPLET: INCORPORATING A TIMER

This third applet is an attempt to make the applet-student interaction more “game-like,” to increase student engagement. The student is given four rectangular-coordinate pairs and four points, and he/she must drag the four points to their proper locations. See Figure 3 and Figure 4. A timer is included to challenge the student to place the four points as quickly as possible. The student clicks a button to reset the applet and start the timer; after all the points have been placed correctly, the student clicks the button again and his/her time is displayed. The JavaScript code keeps track of the “best time” and the student is challenged to keep getting lower and lower times.

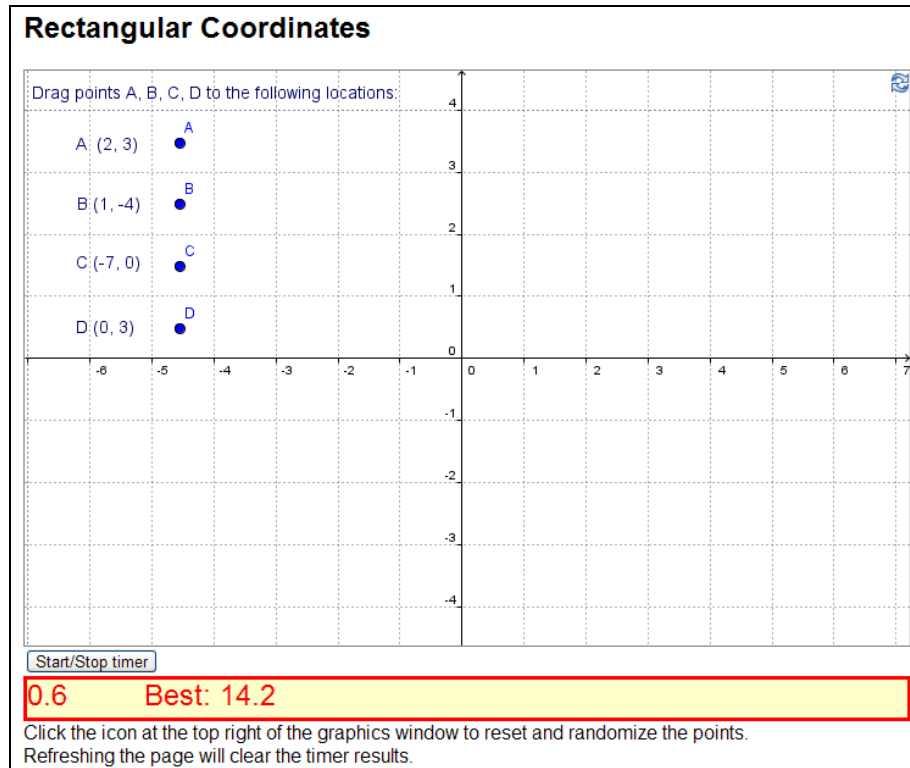


Figure 3: A new challenge has just started and the timer is running.

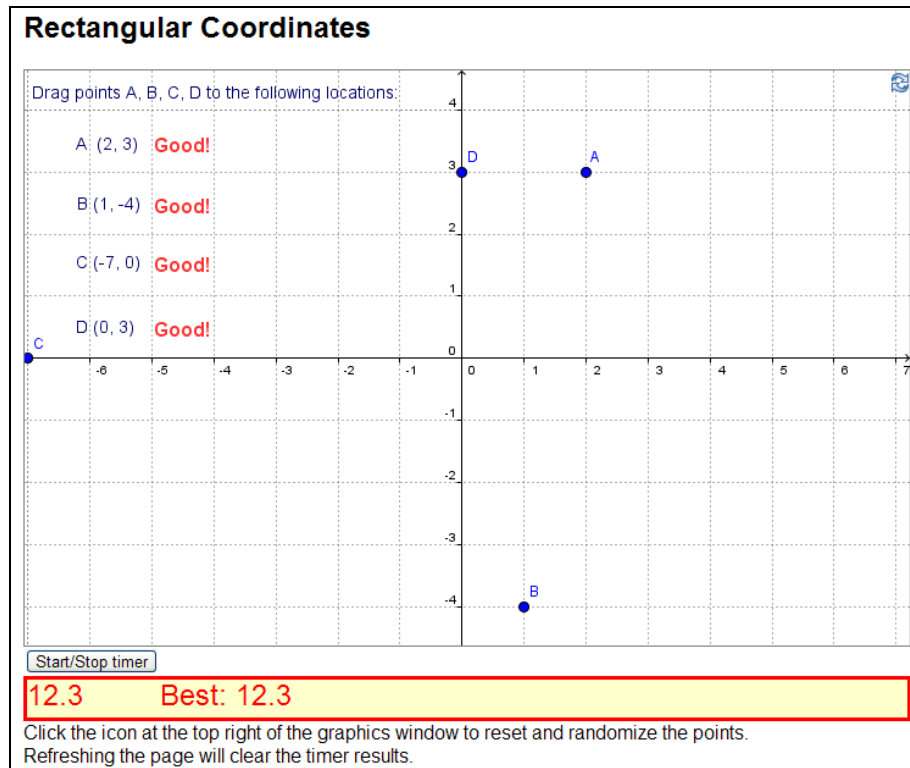


Figure 4: After the points have been placed and the Start/Stop button has been clicked

The dynamic text below the Start/Stop button is achieved using a `<div>` html element.

5. SUMMARY

Using GeoGebra and JavaScript, one can create simple, engaging web-based applets. The first and the third applets presented algebraic representations of an object and the student had to manipulate the graphics to match it. In the second applet, the graphical representation was given and the student had to type an algebraic representation.

In the first applet, JavaScript was incorporated only to re-randomize the applet and start a new problem. In the second applet its role was increased as it modified the student's answer and submitted that answer to GeoGebra to check for correctness. Finally, in the third applet, JavaScript was used to create a timer that gave the applet a more game-like feel.

GeoGebra and JavaScript together provide a powerful tool in creating web-based applets that engage students.