Unit 5: Lists, Graphics, and Dynamic Programs | **Application: The Sierpinski Gasket**

In this application, we will develop a project to play the 'Chaos Game' that generates the Sierpinski Gasket, a famous fractal.

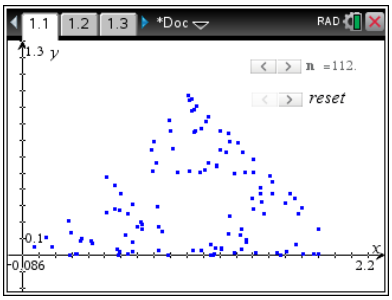
Objectives:

- Write a program that generates an interesting image (the Sierpinski Gasket)
- Control the permitted actions allowed on a slider

Teacher Tip: This project generates a FRACTAL. Key mathematics concepts are: basic coordinate geometry and especially the midpoint formula. **Google** Sierpinski Gasket for more information.

The Sierpinski Gasket

The **Sierpinski Gasket**, or **Sierpinski Triangle** is a fractal formed by starting with an equilateral triangle and successively removing the 'center triangles' as in the picture above. This can be done infinitely so the Gasket is infinitely holed and any portion of the image is similar to the image itself (self-similarity).



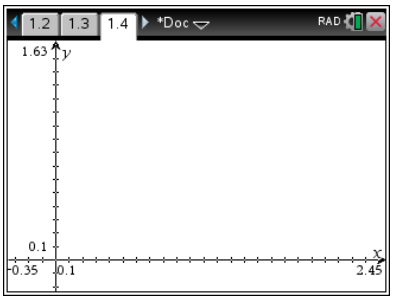
Another way of generating the Gasket that we will use in this project is called the **Chaos Game**. The rules are:

1. Select three (3) points in a plane to form three vertices. For the program described below, the selected points are (0,0), (2,0), and (1,1). (We understand that this is *not* an equilateral triangle.)
2. Start by randomly selecting *any point* (preferably inside the triangle but this does not matter), and consider that point to be your 'current' position.
3. Randomly select any one of the three vertex points.
4. Move half the distance from your current position to the selected vertex (that is, calculate the midpoint of your current position and that vertex).
5. Plot this new current position.
6. Repeat from Step 3.

We will use a growing scatter plot to visually represent this game. Along the way, we'll learn how to **reset** a scatter plot and how to prevent a slider from going backwards (disabling the left arrow or down arrow part of a minimized slider).

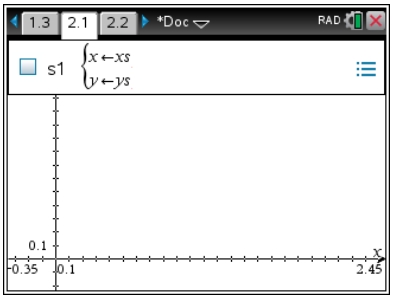
Set Up the Graphs Page

1. Start a new document, and add a **Graphs** app.
2. Move the origin to the lower left corner of the page by "grabbing" the graph at an empty location on the screen. Change the x- and y-axis values so that they are similar to those shown to the right. (Remember: The triangle's vertices will be (0,0), (2,0), and (1,1).) The axes end values are subjective and can be adjusted later.



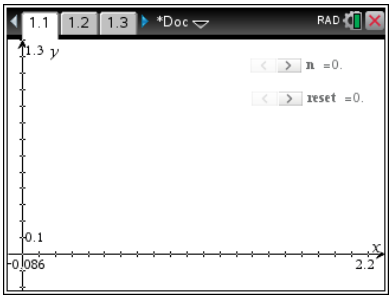
*It's a good idea to save your document now and to save often using **ctrl+S** throughout this project in case something goes awry along the way.*

3. Set up a scatter plot of *xs* and *ys* (**menu > Graph Entry/Edit > Scatter Plot**).



These variables are not yet defined so no graph will appear when you press **enter**. Also, there will be a message related to creating sliders for *xs* and *ys*, but at this point, click on **cancel**.

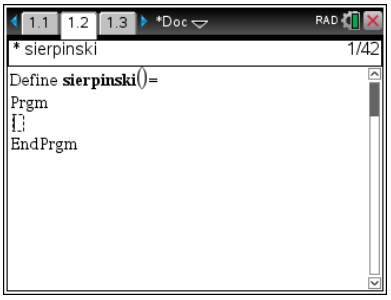
4. Insert *two* sliders (**menu > Actions > Insert Slider**):
   - **n** – value 0, minimum 0, maximum 2000, step size 1, horizontal, and minimized
   - **reset** – value 0, minimum 0, maximum 1, step size 1, horizontal, and minimized



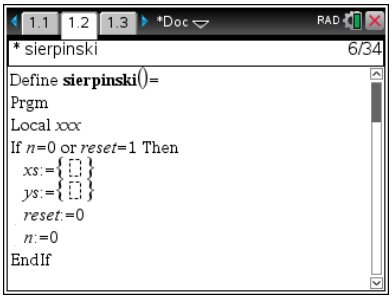
5. Move and place these sliders in the top right corner of the screen so that they do not interfere with the triangle.

Writing the Program

1. Insert a page (**ctrl+doc**), and select **Add Program Editor** from the menu. Select **New…**.
2. Name the program **sierpinski**.
3. Do not add any arguments.



4. Include the **Local** statement here in case any auxiliary variables are needed later in the program. Use the variable **xxx** as a placeholder for now.
5. Enter the **If…Then…EndIf** structure as shown:



If *n*=0 or *reset*=1 **Then**

xs:={ }

ys:={ }

reset:=0

n:=0

EndIF

- The 'reset' actions should occur when the program first runs (n = 0), so we include this condition as part of the first **If** statement.
- When the **reset** slider is used, all the data in lists *xs* and *ys* are erased (assigned an empty set), and the variables *reset* and *n* are both set back to 0.
- Setting *reset*:=0 in this part of the program might seem like an unusual choice, but the result is that the value of **reset** on the screen will *always appear* to be 0.
- Clicking the **reset** button causes **reset** to become 1, forcing the program to run. The program immediately sets the value of **reset** back to 0 (along with the other commands in this part of the program).

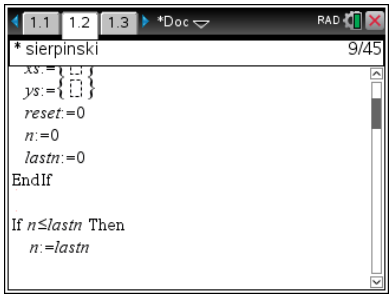
We want the value of **n** (the number of points plotted) to increase and not decrease. Do this by keeping track of the last (previous) value of **n** and compare it to the new (current) value of **n**. If the new value is less than the last value, it will be ignored by setting **n** to be the last value.

6. Use the variable **lastn** to hold the previous value of **n**. This should be entered before **EndPrgm**:

   lastn:= n
   EndPrgm

7. Also, initialize **lastn** in the first **If** structure:

   lastn:= 0



The next portion of code checks to see if n is less than the previous n (when the left arrow of the **n** slider is clicked).

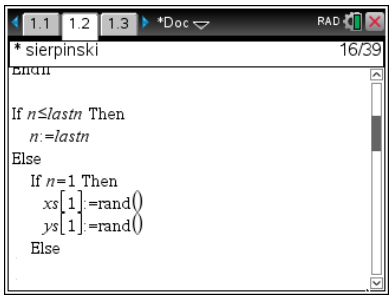
8. After the initializing section, add:

   If n < lastn Then
   n:= lastn
   Else

9. Next, after the **Else** statement, build the lists for the scatter plot.

   By the rules of the Chaos Game (rule 2), there first needs to be a random number selected. The first data point can be any point. So, if n is 1, we then assign a random number to xs[1] and ys[1]:

   If n = 1 Then
   xs[1]:= rand()
   ys[1]:= rand()
   Else



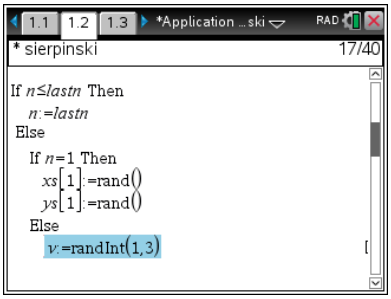
Recall that the **rand()** function generates a random decimal from 0 to 1. The x- and y-coordinates will both have a value between 0 and 1.

We're now ready to tackle the heart of the algorithm mentioned at the start. We'll repeat it here to refresh our memory of these rules from the **Chaos Game**:

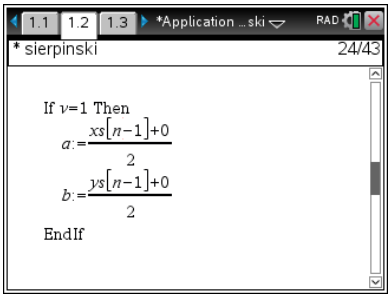
3. Randomly select any one of the three vertex points.
4. Move half the distance from your current position to the selected vertex (that is, calculate the midpoint of your current position and that vertex).
5. Plot this new current position. (Note that this new position is added to lists *xs* and *ys*.)
6. Repeat from Step 3.

10. According to rule 3 of the Chaos Game, now randomly select one of the vertices of the original triangle.

    Use **v:= randint(1,3)** to select a random integer which will help distinguish among the three vertices.

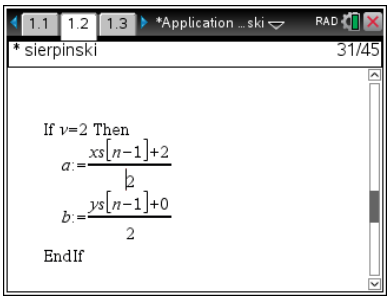


Build a set of If statements based on v to compute the next midpoint. Recall that the three vertices we've chosen to use are (0, 0), (2, 0), and (1, 1).



11. If **v**=1, arbitrarily use the point (0, 0) and the last point in the lists to calculate a midpoint.

    Recall from Geometry that the coordinates of the midpoint between two points (x1, x2) and (y1, y2) are (x1+x2)/2 and (y1+y2)/2.

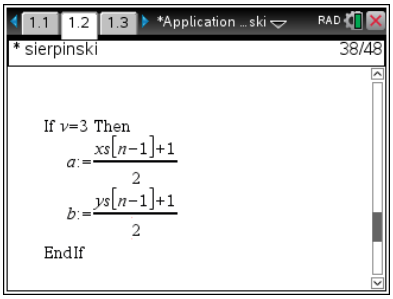


At this location in the program, the last values of **xs** and **ys** are **xs[n-1]** and **ys[n-1]**; we got here by increasing **n** but have not yet added the coordinates of another point to the lists.

This analysis gives rise to the code seen to the right. **v**, **a,** and **b** are temporary local variables.

12. At the top of the program, edit the Local statement to become:

    Local v, a, b

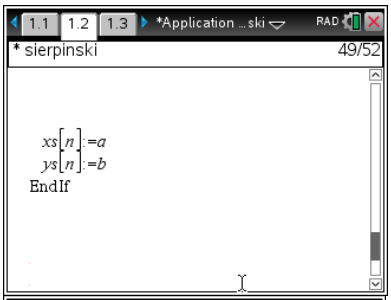


13. We can then easily duplicate this **If** structure to handle the other two vertices, (2,0) and (1,1).

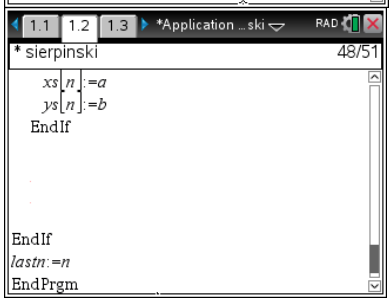
Note: A more efficient structure here would be the **If…Then…ElseIf…Else…EndIf** structure. See if you can implement this section of the program using that structure instead.

14. Add the values of **a** and **b** to the end of our two lists.

    The n^{th} position is actually one spot past the end of the lists, and this is always permitted.



If you've been selecting the **If** structures from the Control menu and inserting the blocks in the correct places, then you'll have another EndIf at the bottom of the program just before **lastn:=n**.



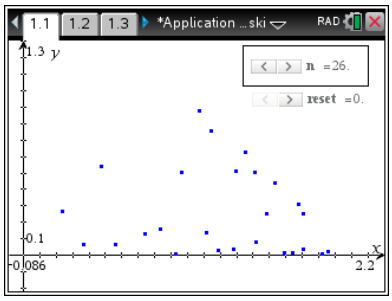
Running the Program

1. Press **ctrl+B** to 'Check Syntax & Store.'

   If there are any errors, you must check your code carefully. A complete program listing is provided at the end of this document.

2. Add a **Notes** app, and insert a **Math Box** (**ctrl+M**). Type the name of the program, a left parenthesis and press **enter**.

3. Switch to the **Graphs** app, and hide the **(xs,ys)** label. Try the two sliders.
   - **n** adds points to the scatter plot. **reset** will erase all points and set **n** back to 0.
   - The left arrow button on the **n** slider should not work, and the left arrow button on the **reset** slider should always be disabled.
   - **reset** should always appear to be 0 because the program detects its change to 1 and immediately changes it back to 0.
   - The value of **n** is limited by the maximum value set in the slider's Settings.
   - The more points plotted, the closer the image appears to represent the Sierpinski Gasket.



Congratulations! Be sure to save and share your accomplishments!

4. Now back to the Local statement. Which variables in this program can be Local? Which variables are needed to produce the picture?