**Lektion 5: Listen, Graphiken und Dynamik** | **Anwendung: Das Sierpinski-Dreieck**

In dieser Anwendung werden wir ein Projekt entwickeln, um das 'Chaos-Spiel' zu spielen, bei dem das Sierpinski-Dreieck, ein berühmtes Fraktal, entsteht.

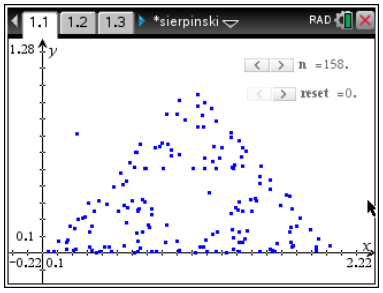
**Lernziele:**

- Ein Programm schreiben, das ein interessantes Bild erzeugt (das Sierpinski-Dreieck)
- Alle erlaubten Aktionen über Schieberegler kontrollieren

**Das Sierpinski-Dreieck**

Das **Sierpinski-Dreieck**, oder auch die „**Sierpinski-Dichtung**" genannt, ist ein Fraktal, das mit einem gleichseitigen Dreieck beginnt, wobei schrittweise die 'inneren Dreiecke' entfernt werden wie oben abgebildet. Das kann unendlich oft fortgesetzt werden bis das Dreieck 'unendlich ausgehöhlt' ist und jeder Teil des Bilds ähnlich zum ganzen Bild wird (Prinzip der Selbstähnlichkeit).

In diesem Projekt werden wir diese Figur auf eine andere Weise herstellen, nämlich mit dem so genannten **Chaos-Spiel**. Dieses Spiel hat die folgenden Regeln:



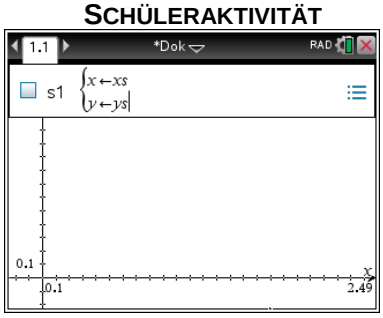

1. Wähle drei Punkte in der Ebene als Ecken eines Dreiecks: (0,0), (2,0) und (1,1). Wir wissen, dass dies kein gleichseitiges Dreieck ist.
2. Starte mit der Wahl eines Zufallspunkts (vorzugsweise innerhalb des Dreiecks, das muss aber nicht sein) und betrachte diesen Punkt als deine 'momentane' Position.
3. Wähle zufällig einen der drei Eckpunkte des Dreiecks aus.
4. Bewege dich auf halbem Weg von deiner Position zur gewählten Ecke, d.h., zum Mittelpunkt der Strecke von dir zu diesem Eckpunkt.
5. Zeichne einen Punkt an diese Stelle.
6. Wiederhole die Prozedur ab Schritt 3.

Wir werden ein wachsendes Streudiagramm zur Visualisierung dieses Spiels herstellen. Dabei werden wir lernen, wie man ein Streudiagramm zurücksetzt (ein Reset macht) und wie man den Schieberegler am Zurückgehen hindern kann (indem man den Pfeil-Links oder Pfeil-Rechts-Teil des verkleinerten Schiebereglers ausschaltet).

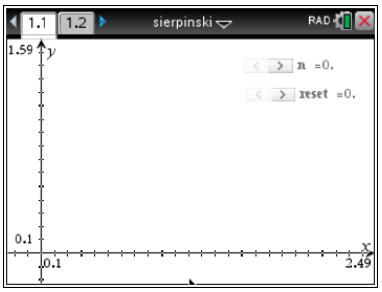
1. Beginne mit einem neuen Dokument und füge eine **Graphs**-Seite ein.
2. Bewege den Koordinatenursprung in die linke untere Ecke der Seite und strecke die Achsen so, dass die x-Achse bequem von 0 bis 2 und die y-Achse von 0 bis etwas mehr als 1 reicht. (Denke an die Koordinaten der Dreiecksecken.)
3. Verziehe die Koordinatenebene um den Ursprung zu bewegen und ziehe an der x-Achse, um die Skalierung zu ändern. Das kann auch noch später angepasst werden.


***Es ist immer gut, das Dokument regelmäßig zu speichern (mit ctrl+S), weil ja immer wieder etwas schiefgehen kann.***

4. Richte ein Streudiagramm für die Listen *xs* und *ys* ein (***menu> Graph Eingabe/Bearbeitung> Streudiagramm***).
   - Diese Variablen sind noch nicht definiert, daher ist auch nach einem enter kein Diagramm zu sehen.



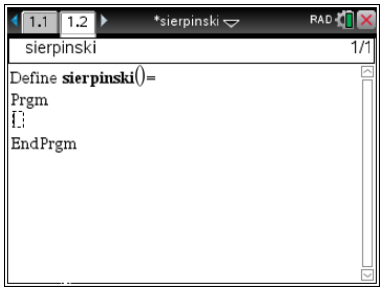

5. Füge *zwei* Schieberegler ein (***menu> Aktionen> Schieberegler einfügen***):
   - **n** – Anfangswert 0, Minimum 0, Maximum 2000, Schrittweite 1, horizontal und minimiert.
   - **reset** – Anfangswert 0, Minimum 0, Maximum 1, Schrittweite 1, horizontal und minimiert.
6. Platziere diese Regler in die obere rechte Ecke des Schirms, so dass sie nicht das Dreieck teilweise überdecken.



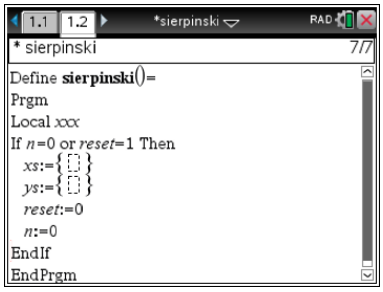

In der Graphs-Seite sind wir fertig.

**Das Programm**

1. Füge eine weitere Seite für den Programmeditor ein (***doc> Einfügen > Programmeditor > Neu …***).
2. Nenne das Programm **sierpinski**.




3. Füge hier auch eine Local-Anweisung für den Fall ein, dass später im Programm Hilfsvariable benötigt werden.
   - Verwende die Variable **xxx** vorläufig als Platzhalter.
4. Beginne den Programmcode mit dem ‘reset’-Konzept:
   - Wenn der **reset**-Schieberegler verwendet wird, werden alle Daten in den Listen *xs* und *ys* gelöscht und die Variablen **reset** und **n** auf 0 gesetzt.
   - Wenn das Programm zum ersten Mal läuft, wird es initialisiert (wenn n = 0 und reset ≠ 1).




Die Zuweisung **reset:=0** in diesem Programmteil mag seltsam anmuten, aber es bewirkt, dass der Wert von **reset** am Schirm immer als 0 erscheinen wird.

Beim Klick auf die **reset**-Schaltfläche nimmt **reset** den Wert 1 an und veranlasst einen Programmlauf. Dann wird **reset** sofort wieder auf 0 zurückgesetzt (gemeinsam mit anderen Werten in diesem Programmteil).

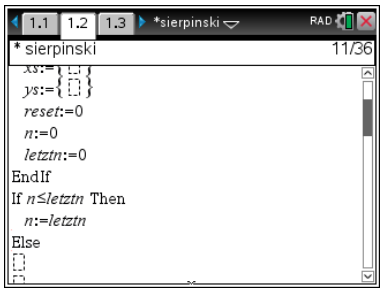
**Wie verhindern wir n am Abnehmen?**

Wir wünschen, dass der Wert von **n** (die Anzahl der zu zeichnenden Punkte) nur zu- aber nicht abnimmt. Zu diesem Zweck merken wir uns den jeweils letzten Wert von n und vergleichen ihn mit dem allfällig neuen Wert. Ist dieser neue Wert kleiner als der neue, dann wird dieser ignoriert und **n** wieder auf den letzten Wert gesetzt.

1. Mit der Variablen **letztn** wird der letzte Wert von **n** gehalten. Das geschieht ganz am Programmende:

```
letztn := n
EndPrgm
```

Im nächsten Programmteil wird geprüft, ob das aktuelle **n** kleiner ist als das vorige **n** (wenn der Linkspfeil am Schieberegler angeklickt wird)?

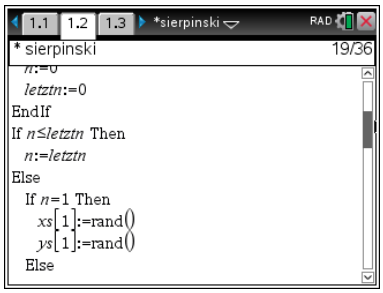



2. Füge nach dem Initialisierungsteil hinzu:

```
If n < letztn Then
    n := letztn
Else
```

3. Gib **letztn** auch einen Anfangswert in der ersten **If**-Konstruktion:

```
letztn:=0
```




4. Jetzt werden die Listen für das Streudiagramm gefüllt:
   - Der erste Punkt ist etwas Besonderes, denn das kann ein beliebiger Punkt sein. Wenn also n=1, belegen wir *xs*[1] und *ys*[1] mit einer Zufallszahl:

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

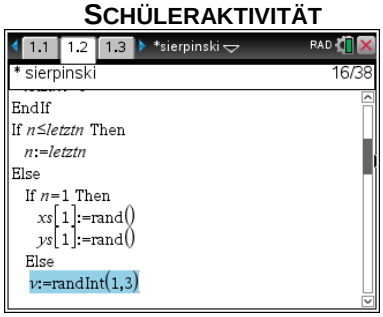
Die rand()-Funktion erzeugt eine Zufallszahl zwischen 0 und 1. Damit ist dieser Punkt tatsächlich ein Zufallspunkt im Quadrat zwischen (0,0) und (1,1).

Jetzt sind wir bereit, den Kern des zu Beginn erklärten Algorithmus anzugehen. Wir holen ihn kurz ins Gedächtnis zurück:

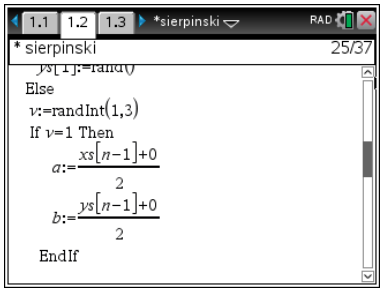
3. Wähle zufällig einen der drei Eckpunkte des Dreiecks aus.
4. Bewege dich auf halbem Weg von deiner Position zur gewählten Ecke, d.h., zum Mittelpunkt der Strecke von dir zu diesem Eckpunkt.
5. Zeichne einen Punkt an diese Stelle.

Wiederhole den Prozess ab Schritt 3.

5. Mit **v := randint(1,3)** wählen wir eine ganzzahlige Zufallszahl, die helfen wird, zwischen den drei Eckpunkten zu unterscheiden.
6. Konstruiere eine Reihe von If-Anweisungen auf der Grundlage von v um den nächsten Mittelpunkt zu berechnen.
   - Erinnere dich: unsere drei Eckpunkte sind (0, 0), (2, 0) und (1, 1).




7. Wenn **v**=1, verwende den Punkt (0, 0) und den zuletzt gefundenen Punkt, um einen Mittelpunkt zu berechnen.
   - Erinnere dich an die Geometrie, wie man den Mittelpunkt einer Strecke zwischen zwei Punkten (x1,x2) und (y1,y2) findet: (x1+x2)/2 und (y1+y2)/2.



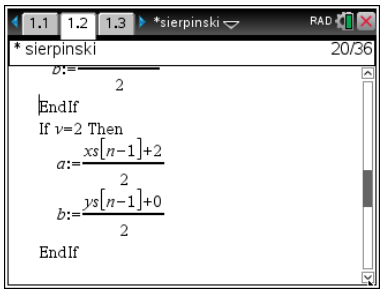

An dieser Programmstelle: die letzten Werte von **xs** und **ys** sind **xs[n-1]** und **ys[n-1]**, denn wir sind hierher durch Erhöhung von **n** gekommen, ohne ein weiteres Element zur Liste hinzugefügt zu haben.

Das erklärt den rechts abgebildeten Code. **v**, **a** and **b** sind temporäre lokale Variable.

8. Füge am Kopf des Programms die Anweisung

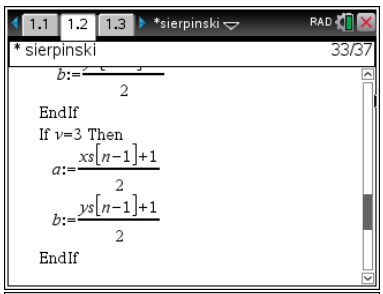
   **Local v, a, b**

   ein.

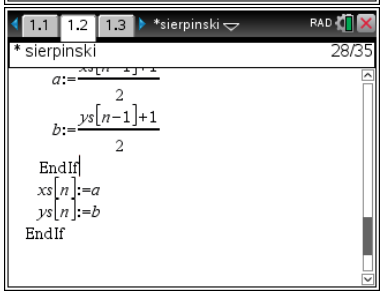



Diese **If**-Konstruktion lässt sich leicht kopieren und damit werden die beiden anderen Ecken, (2,0) und (1,1) angesprochen.

Anmerkung: Eine effektivere Konstruktion wäre an dieser Stelle ein If…Then…ElseIf…Else…EndIf. Versuche, ob du diesen Programmteil damit schreiben könntest.

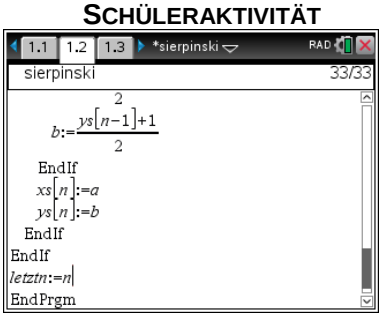



9. Hänge die Werte von **a** und **b** ans Ende der beiden Listen dran.
   - die n-te Position ist immer eine Stelle nach dem Ende der Listen, und das ist immer erlaubt.



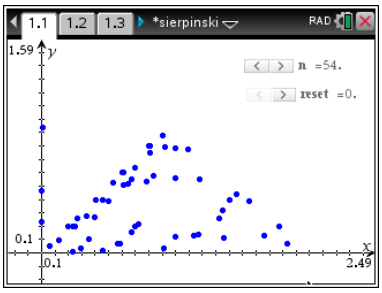
Wenn du die **If**-Anweisungen aus dem Steuerungsmenu an die richtigen Stellen eingefügt hast, dann wirst du jetzt ein weiteres **EndIf** am Programmende gerade noch vor **letztn := n** vorfinden.

10. Mit ctrl B überprüfe die Syntax des Programms und speichere es.
    - Falls Fehlermeldungen auftreten, überprüfe sorgfältig den Code. Ein vollständiges Programmlisting findest du in der Lehrerinformation.
11. Füge eine **Notes**-Seite ein, erzeuge eine **Math Box** (ctrl M) und tippe den Programmnamen, eine like Klammer und enter.





12. Wechsle in die **Graphs**-Seite und bewege die Schieberegler.
    - **n** fügt Punkte zum Streudiagramm hinzu und **reset** löscht alle Punkte und setzt **n** zurück auf 0.
    - Der Linkspfeil am Schieberegler für **n** soll nicht wirksam sein. Der Linkspfeil am **reset**-Schieberegler ist immer ausgeschaltet.
    - **reset** sollte immer mit dem Wert 0 erscheinen, denn wenn das Programm einen Wechsel auf 1 entdeckt, wird sofort auf 0 zurückgesetzt.
    - Der Wert für **n** hat seine Grenze im Wert für Maximum in den Einstellungen des Schiebereglers.
    - Je mehr Punkte du zeichnen lässt, desto mehr wird das Bild dem Sierpinski-Dreieck ähnlich werden.




Bravo und Glückwunsch!