Unit 5: Lijsten, plaatjes (graphics) en dynamische programma's

Toepassing: De zeef van Sierpinski

In deze toepassing ontwikkelen we een project om het 'chaos spel' te spelen dat de zeef van Sierpinski, een beroemde fractal, genereert.

Doelen:

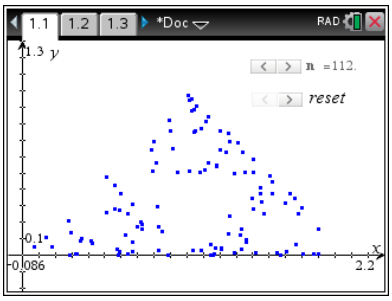
- Een programma schrijven dat een interessant plaatje genereert (de zeef van Sierpinski)
- De toegestane acties op een schuifknop besturen

Docenten Tip: Dit project genereert een FRACTAL. Belangrijke wiskundige concepten zijn: basale coördinatenmeetkunde en in het bijzonder de middelpuntsformule. **Google** op (de zeef) van Sierpinski voor meer informatie.

De zeef van Sierpinski

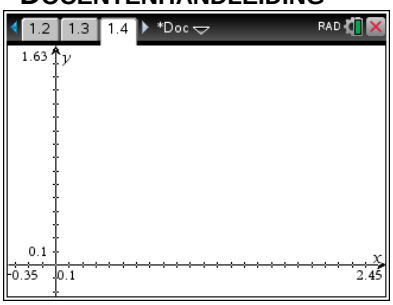
De **zeef (of driehoek) van Sierpinski** is een fractal die wordt gevormd door te beginnen met een gelijkzijdige driehoek en achtereenvolgens de 'middelste' driehoeken te verwijderen zoals in de afbeelding hierboven te zien is. Dit kan oneindig vaak gedaan worden, zodat de de zeef oneindig veel gaten heeft en elk gedeelte van de afbeelding gelijk is aan de afbeelding zelf (zelfgelijkheid).

Een andere manier om de zeef te genereren, die we in dit project zullen gebruiken, heet het **Chaos spel**, waarbij dit de regels zijn:



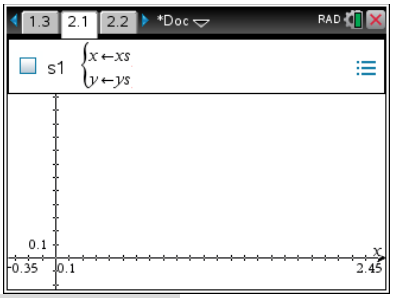
1. Kies drie (3) punten in een vlak om drie hoekpunten te vormen (0,0), (2,0), en (1,1). We snappen dat dit *geen* gelijkzijdige driehoek is.
2. Begin met het kiezen van een *willekeurig punt* (bij voorkeur binnen de driehoek, maar dat maakt eigenlijk niet uit) en beschouw dat punt als je huidige' positie'.
3. Kies willekeurig een van de drie hoekpunten.
4. Beweeg de helft van de afstand tussen je huidige positie en het geselecteerde hoekpunt in de richting van dit hoekpunt. (Dat wil zeggen: bereken het middelpunt van het lijnstuk tussen je huidige positie en dat hoekpunt).
5. Plot deze nieuwe 'huidige' positie.
6. Herhaal dit proces vanaf stap 3.

We zullen een groeiende puntenwolk gebruiken om dit spel visueel te maken. Tijdens dit project zullen we leren hoe we een puntenwolk opnieuw kunnen instellen ('**resetten')** en hoe we voorkomen dat een schuifknop 'terugloopt' (door het gedeelte met het pijltje naar links of naar beneden van een geminimaliseerde schuifknop ui te schakelen).



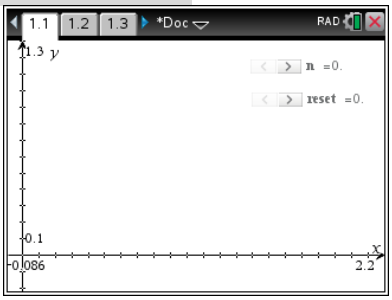
1. Begin met een nieuw document en voeg een toepassing **Grafieken** toe.
2. Verplaats de oorsprong naar de linkeronderhoek van de pagina en rek de assen uit zodat de x-as loopt van 0 tot ruim 2 en de y-as van 0 tot iets boven 1 [omdat de hoekpunten van onze driehoek (0,0), (2,0) en (1,1) zijn].
3. Pak de grafiek ergens vast om de oorsprong te verplaatsen en pak de assen vast om de schaal op de x-as uit te rekken. Dit is subjectief en kan later worden aangepast.

   Het is altijd een goed idee om je document nu te bewaren en het regelmatig op te slaan (gebruik crtl-s) tijdens dit project voor het geval er onderweg iets mis gaat.



4. Stel een puntenwolk in van *xs* en *ys* (**menu > Grafiek invoeren/bewerken > Puntenwolk**).
   - Deze variabelen zijn nog niet gedefinieerd dus er zal nog geen grafiek verschijnen wanneer je op [enter] drukt.

Docenten Tip: Het is een goed idee om deze activiteit eerst op papier uit te proberen. Gebruik een dobbelsteen om een willekeurig hoekpunt te selecteren en vind de plek van het midden tussen het huidige punt en dat hoekpunt. Herhaal dit.

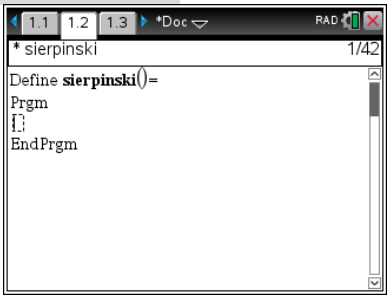


5. Voeg *twee* schuifknoppen in (**menu > Acties > Schuifknop invoegen**):
   - **n** – waarde 0, minimum 0, maximum 2000, stapgrootte 1, horizontaal en geminimaliseerd.
   - **reset** – waarde 0, minimum 0, maximum 1, stapgrootte 1, horizontaal en geminimaliseerd.
6. Plaats deze schuifknoppen in de rechterbovenhoek van het scherm zodat ze de driehoek niet in de weg zitten.

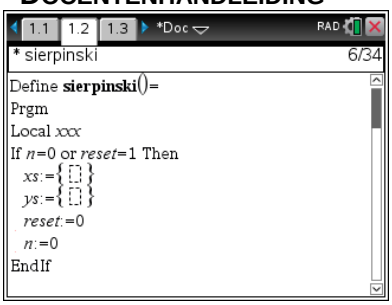
Je bent nu klaar met de toepassing Grafieken.

Docenten Tip: schuifknoppen kunnen later worden aangepast (bewerkt en verplaatst). De variabelen die gebruikt zijn in de schuifknoppen zullen ook in het programma worden gebruikt..

Programmeren



1. Voeg een nieuwe pagina toe (ctrl-doc) en voeg een programma-editor in (**doc > Invoegen > Programma Editor > Nieuw…**).
2. Geef het programma de naam **sierpinski**.



3. Voeg hier de opdracht Local toe voor het geval er later nog hulpvariabelen nodig zijn in het programma.
   - Gebruik de variabele **xxx** als een tijdelijke plaatsvervanger.
4. Begin je programmacode met het concept '**reset'**.
   - Wanneer de schuifknop **reset** wordt gebruikt, wis je alle gegevens in de lijsten *xs* en *ys* en stel je de variabelen **reset** en **n** opnieuw in op 0.
   - Initialiseer deze ook voor de eerste keer dat je het programma probeert uit te voeren (wanneer **n** is 0) (en reset is niet 1).

Het instellen van **reset:=0** in dit gedeelte van het programma lijkt misschien een vreemde keuze maar het resultaat is dat de waarde van **reset** op het scherm altijd 0 lijkt te zijn.

Klikken op de knop **reset** zorgt ervoor dat **reset** 1 wordt, hetgeen de uitvoering van het programma forceert. Vervolgens zet het programma de waarde van **reset** onmiddellijk terug op 0 (tegelijk met de andere acties in dit deel van het programma).

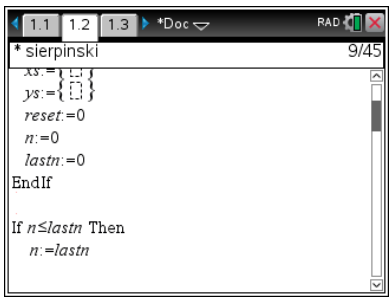
> **Docenten Tip:** het idee van het besturen van de waarden van de variabelen van een schuifknop is behoorlijk geraffineerd, maar het is hier nodig omdat we niet willen dat de punten in de driehoek (de zeef) verdwijnen.

Voorkomen dat de waarde van n afneemt

We willen dat de waarde van **n** (het aantal punten dat geplot wordt) toeneemt en niet afneemt. Doe dit door de *laatste (vorige) waarde van* **n** bij te houden en deze te vergelijken met de nieuwe (huidige) waarde van **n**. Als de nieuwe waarde kleiner is dan de laatste (vorige) waarde, dan wordt deze genegeerd door n in te stellen op de laatste (vorige) waarde.

1. Gebruik de variabele **lastn** om de vorige waarde van **n** in op te slaan. Dit gebeurt helemaal aan het eind van het programma:

```
lastn := n
EndPrgm
```



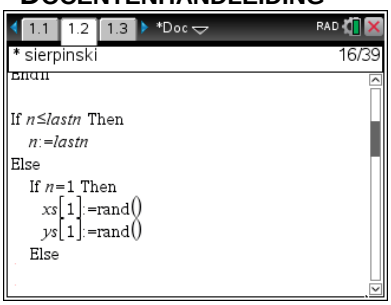
Het volgende deel van het programma gaat na of n kleiner is dan de vorige n (wanneer het pijltje naar links op de schuifknop is aangeklikt).

2. Voeg na het programmagedeelte voor het initialiseren, toe:

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

3. Initialiseer **lastn** ook in de eerste **If** structuur:

```
lastn:=0
```



4. Bouw vervolgens de lijsten voor de puntenwolk op.
   - Het eerste punt is speciaal omdat dit elk willekeurig punt kan zijn. Dus, als n is 1, dan kennen we een toevalsgetal toe aan xs[1] en ys[1]:

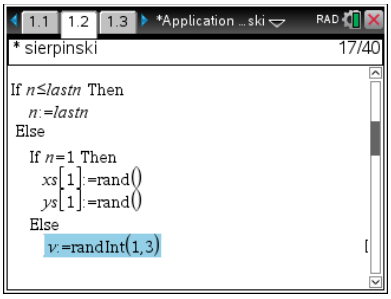
```
Else
  If n = 1 Then
    xs[1] := rand()
    ys[1] := rand()
  Else
```

De functie rand( ) genereert een willekeurig decimaal getal tussen 0 en 1, dus in feite is dit punt een willekeurig punt in het vierkant tussen (0,0) en (1,1).

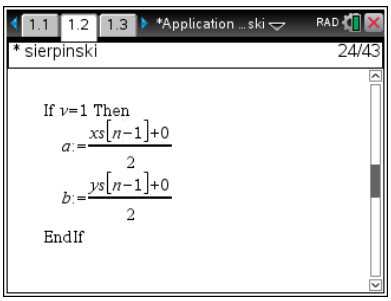
We zijn nu klaar om de kern van het algoritme dat aan het begin werd beschreven te gaan aanpakken. We herhalen het hier om ons geheugen op te frissen:

5. Kies willekeurig een van de drie hoekpunten.
6. Beweeg de helft van de afstand tussen je huidige positie en het geselecteerde hoekpunt in de richting van dit hoekpunt (dat wil zeggen: bereken het middelpunt van het lijnstuk tussen je huidige positie en dat hoekpunt).
7. Plot deze nieuwe huidige positie.

Herhaal vanaf stap 3.



8. Gebruik **v := randint(1,3)** om een willekeurig geheel getal te selecteren, dit helpt om de drie hoekpunten te onderscheiden.
9. Bouw een serie opdrachten gebaseerd op v om het volgende midden (middelpunt) te berekenen.
   - Herinner je dat de hoekpunten die we hebben gekozen (0, 0), (2, 0) en (1, 1) zijn.



10. Als **v**=1 gebruik dan (willekeurig) het punt (0, 0) en het laatste punt in de lijst om het midden te berekenen.
    - Herinner je uit de meetkundelessen dat de coördinaten van het midden tussen twee punten (x1, x2) en (y1, y2) gelijk zijn aan (x1+x2)/2 en (y1+y2)/2.

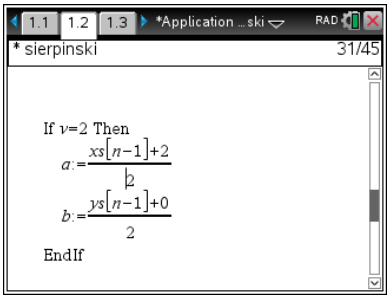
Op deze plek in het programma zijn de laatste waarden van **xs** en **ys** gelijk aan **xs[n-1]** en **ys[n-1]** omdat we hier zijn gekomen door **n** toe te laten nemen, maar we nog niet een nieuw punten hebben toegevoegd aan de lijsten.

Deze analyse leidt tor de programmacode die je hier rechts ziet. **v**, **a** en **b** zijn tijdelijke, lokale variabelen.

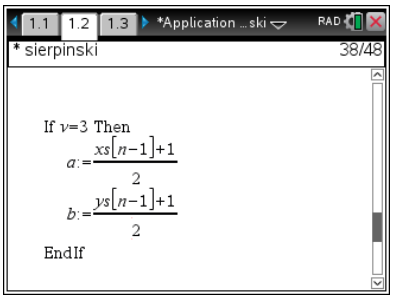
11. Bewerk bovenin het programma de opdracht Local zodat deze wordt:

Local v, a, b

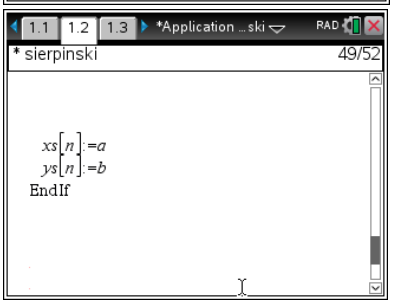
We kunnen dan eenvoudig deze **If**-structuur dupliceren om de andere twee hoekpunten (dit zijn (2,0) en (1,1)) te behandelen.



Opmerking: Hier zou de If…Then…ElseIf…Else…EndIf –structuur een efficiëntere structuur zijn. Bekijk of je dit gedeelte van het programma kunt schrijven met deze structuur.

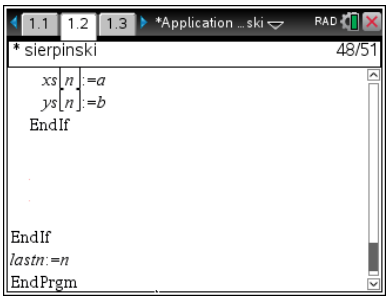


12. Voeg de waarden van **a** en **b** toe aan het eind van onze twee lijsten.
    - De n[de] positie is in feite een plek voorbij het eind van de lijsten en dit is altijd toegestaan.

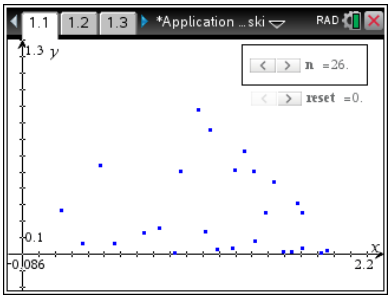


Als je de **If**-structuren hebt geselecteerd uit het menu **Besturing** en je de blokken op de goede plek hebt ingevoegd dan zal je een EndIf onderaan het programma hebben precies voor **lastn:=n**.

13. Druk op **ctrl-B** om '**Syntax controleren en opslaan**' te selecteren.
    - Als er fouten zijn dan moet je je programma zorgvuldig controleren. Het volledige programma is toegevoegd aan het eind van dit document.



14. Voeg een toepassing **Notities** toe; voeg een **wiskundevak** in (ctrl-M); en typ de naam van het programma en een linkerhaakje en druk op [enter].



15. Schakel om naar de toepassing **Grafieken** en test de twee schuifknoppen:
    - **n** voegt punten toe aan de puntenwolk en **reset** zou alle punten moeten wissen en **n** opnieuw instellen op 0.
    - De knop met het pijltje naar links op de schuifknop **n** zou niet moeten werken en het pijltje naar links op de schuifknop **reset** zou altijd uitgeschakeld moeten zijn.
    - **reset** zal altijd 0 lijken te zijn omdat het programma de verandering ervan in 1 opmerkt en dit onmiddellijk terug-verandert in 0.
    - De waarde van **n** wordt begrensd door de maximumwaarde die je hebt ingevoerd in de instellingen voor de schuifknop.
    - Hoe meer punten je plot, hoe beter het plaatje lijkt op de zeef van Sierpinski.

Gefeliciteerd!

Docentenhandleiding:

```
Define sierpinski()=
Prgm
Local v, a, b
If n=0 or reset=1 Then
  xs:={ }
  ys:={ }
  reset:=0
  n:=0
  lastn:=0
EndIf
If n≤lastn Then
   n:=lastn
Else
   If n=1 Then
     xs[1]:=rand()
     ys[1]:=rand()
   Else
      v:=randInt(1,3)
     If v=1 Then
        a:=((xs[n-1]+0)/2)
        b:=((ys[n-1]+0)/2)
     EndIf
     If v=2 Then
        a:=((xs[n-1]+2)/2)
        b:=((ys[n-1]+0)/2)
     EndIf
     If v=3 Then
         a:=((xs[n-1]+1)/2)
         b:=((ys[n-1]+1)/2)
     EndIf
     xs[n]:=a
     ys[n]:=b
   EndIf
EndIf
lastn:=n
EndPrgm
```