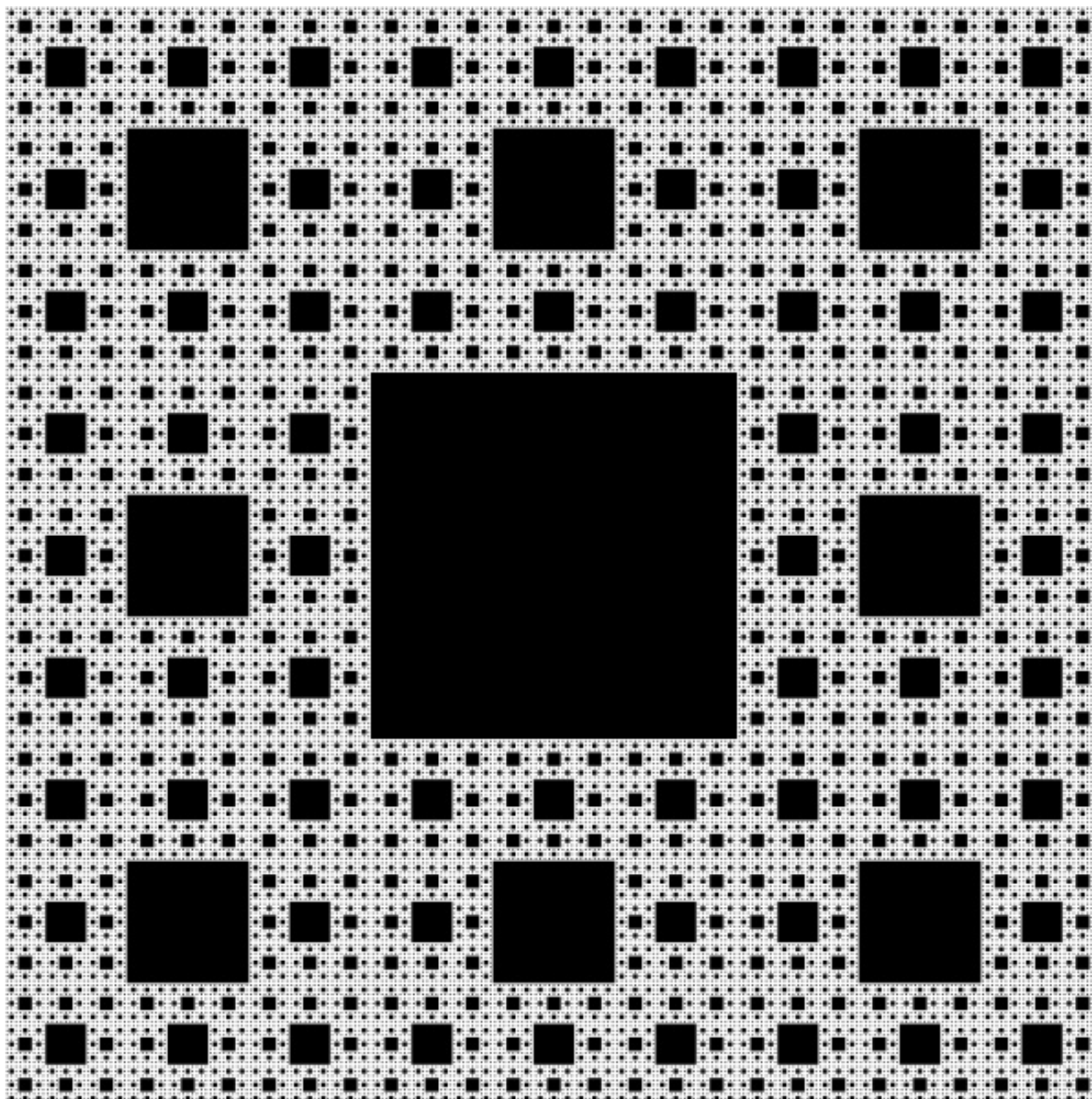




Introduksjon

En fraktal er en geometri med et mønster som gjentar seg selv inne i seg selv. Høres ikke det rart ut? I denne oppgaven skal vi lage våre egne.

Her er Sierpinski-teppet, som er en fraktal:



Steg 1: Hvordan fungerer Sierpinski?

Fraktaler følger tre regler:

- ☐ **Startregelen** gir hvor vi skal starte. Med en firkant? En trekant? En strek?
- ☐ **Tegneregelen** gir hvordan vi skal tegne på nivået vi er. Fargelegge en bit av firkanten? Splitte en strek i to?
- ☐ **Rekursjonsregelen** deler opp figuren vår i mindre biter, som vi kjører på nytt i. Lager firkanten vi tegnet nye firkanter? Lager streken vi tegnet nye streker? Gjenta for hver strek.

☒ Sjekkliste

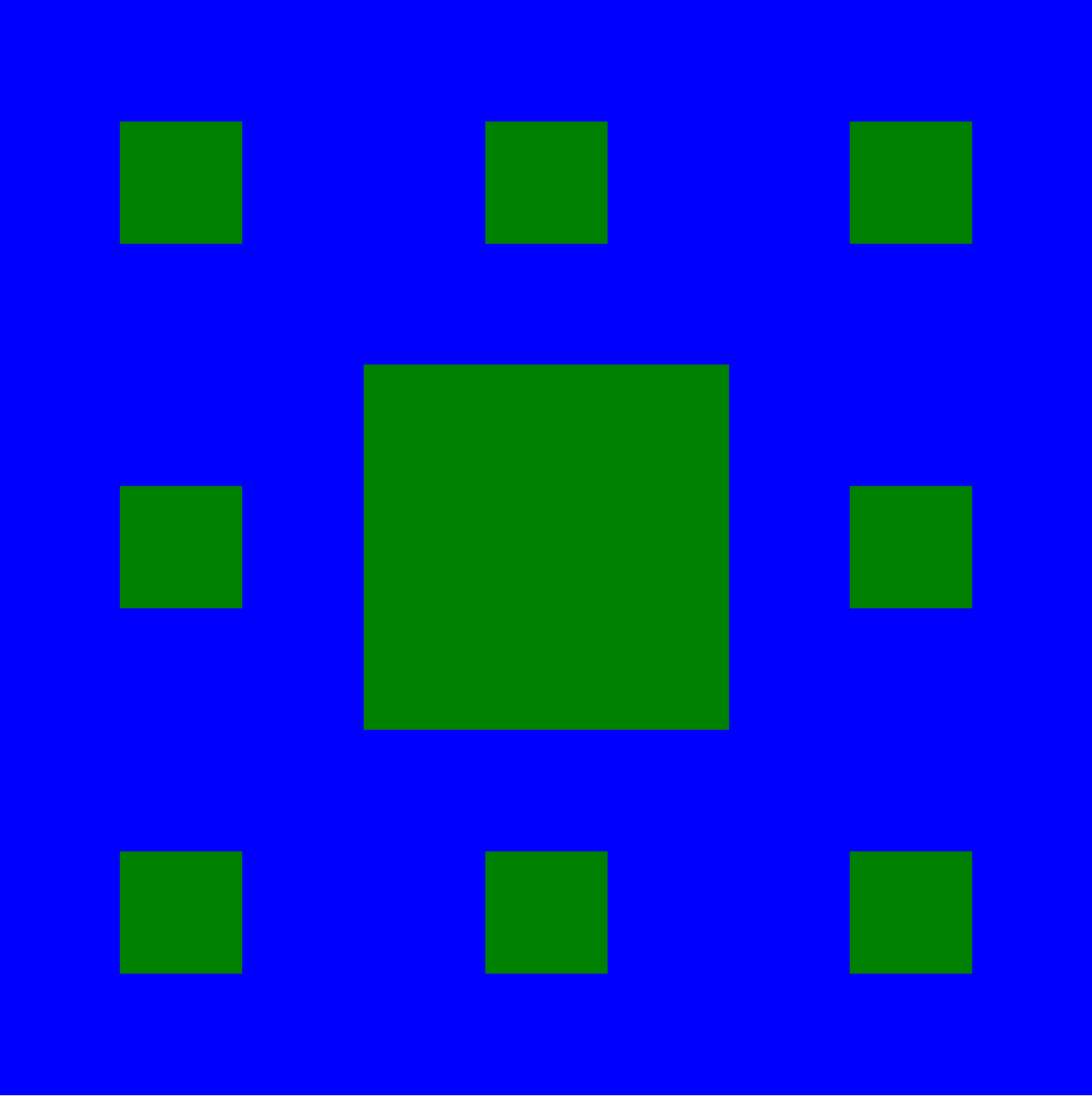
Gå til [Wikipedia-artikkelen](#) til Sierpinski-teppet. Se på animasjonen.

- ☐ Hvordan er teppet før det begynner å bli fargelagt? Dette er **startregelen**.
- ☐ Hva tegner vi i hver firkant? Dette er **tegneregelen**.
- ☐ Hvordan gjentas regelen? Dette er **rekursjonsregelen**.

Se på figurene under avsnittet **Process**. Ser du at noe gjentar seg?

Steg 2: Tegne kvadrater med SVG

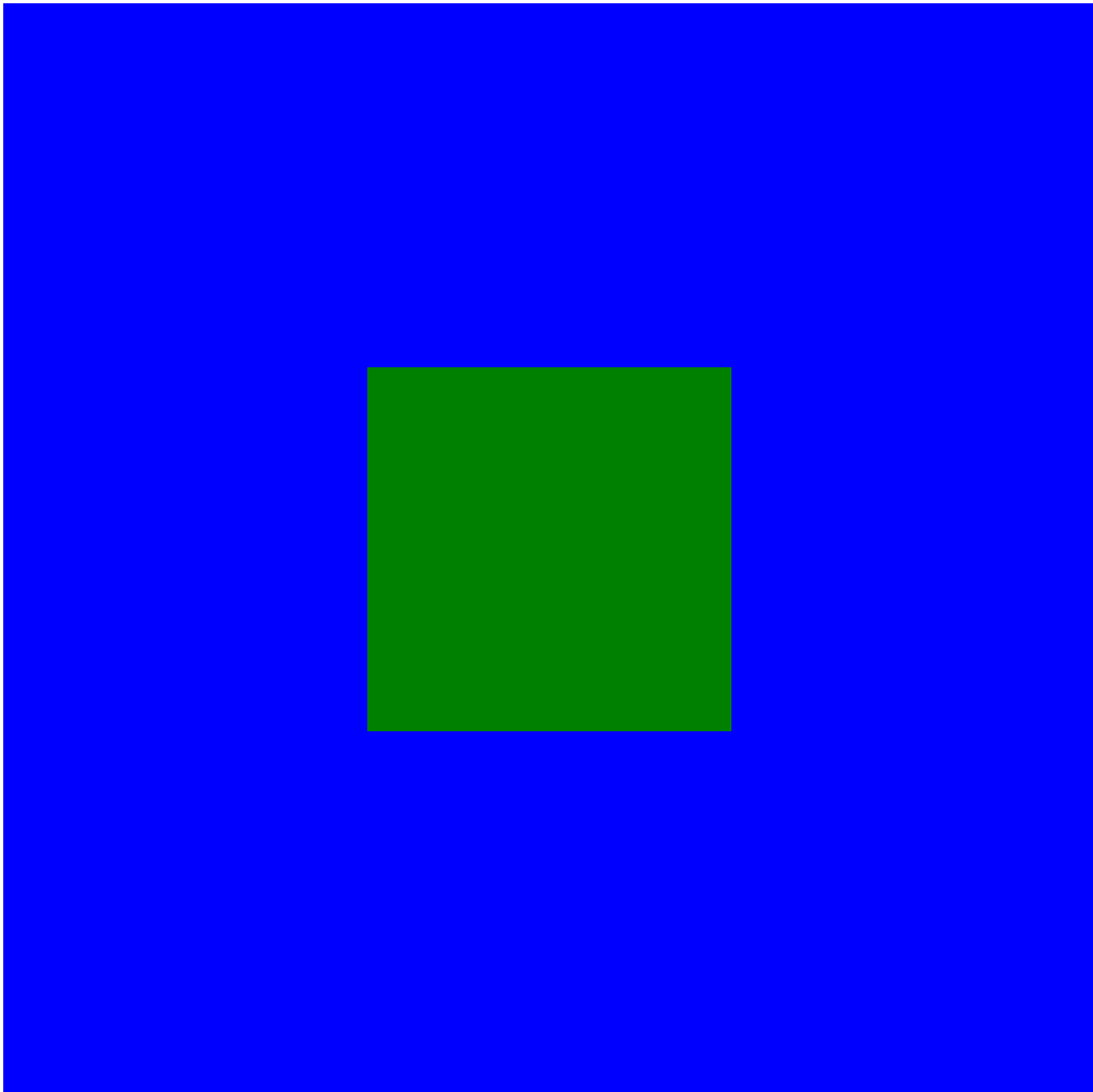
Nå skal vi begynne å tegne kvadratene teppet:



Ett kvadrat kan vi tegne slik:

```
import Svg exposing (svg, rect)
import Svg.Attributes exposing (width, height, viewBox, fill, x, y, width, height)

main =
  svg
    [ width "500", height "500", viewBox "0 0 27 27" ]
    [ rect [ x "0", y "0", width "27", height "27", fill "blue" ] [ ]
      , rect [ x "9", y "9", width "9", height "9", fill "green" ] [ ]
    ]
```



✓ Sjekkliste

- ☐ Hvordan kan vi da tegne mange kvadrater?
- ☐ Hva bestemmer posisjonen til tallene?
- ☐ Hvor mange store grønne kvadrater har du tegnet?
- ☐ Hvor mange små grønne kvadrater har du tegnet?
- ☐ **Utvid koden til å tegne mange kvadrater.**

Steg 3: Datastrukturer

Kan du telle hvor mange kvadrater det finnes i Sierpinski-teppet? Ikke jeg heller. Hmm, gidder vi da å skrive de maaaange linjene SVG for hånd? Nei, vi programmerer!

Vi skal nå representere kvadrater med Records i Elm. Records lar oss lage *våre egne typer*. Vi kommer til å lage en type for punkter og en type for kvadrater.

✓ Sjekkliste

Nå skal du få prøve å lese Elm sine egne lærerressurser.

- ☐ Gå til [Elm-dokumentasjonen for records](#). Finner du eksempelet for et punkt?

Vi legger til en liten snutt i programmet vårt:

```
import Html exposing (div, text, h1)

import Svg exposing (svg, rect)
import Svg.Attributes exposing (width, height, viewBox, fill, x, y, width, height)

myPoint =
  { x = 9
  , y = 3
  }

main =
  div []
```



```
[ h1 [] [ text (toString myPoint) ]
, svg
[ width "500", height "500", viewBox "0 0 27 27" ]
[ rect [ x "0", y "0", width "27", height "27", fill "blue" ] [ ]
, rect [ x "3", y "3", width "3", height "3", fill "green" ] [ ]
]
]
```

Nå kan du endre `toString myPoint` for å skrive ut noe annet.

✓ Sjekkliste

- ☐ Skriv ut kun `x`-attributten til `myPoint`
- ☐ Lag et annet punkt, `yourPoint`. Velg koordinater og skriv ut dette i stedet.
- ☐ Lag et tredje punkt, `theirPoint`. Dette skal du lage *ut ifra* `myPoint`, men du skal bytte ut x-verdien med `0`. Se avsnittet **Updating Records** i guiden.

Nå skal vi ta steget videre og lage våre egne punkter.

Husk! Du kan bruke linjen `[h1 [] [text (toString yz)]` til å teste verdier.

- ☐ Les de to første eksemplene i avsnittet **Record types**.

Her finnes det allerede en `Point`-type vi kan bruke. Har du definert `myPoint` og `yourPoint` på samme måte som det gjøres i guiden?

- ☐ Skriv inn `Point`-typen i programmet ditt
- ☐ Spesifiser at punktene dine skal være av typen `Point`:

```
-- myPoint : Point betyr at myPoint skal være av type Point
myPoint : Point
myPoint = -- din tidligere løsning

-- yourPoint : Point betyr at yourPoint skal være av type Point
yourPoint : Point
yourPoint = -- din tidligere løsning
```

Klager kompilatoren? Hvorfor/hvorfor ikke? Om den klager betyr det ikke at du har gjort noe feil. Det bare at du og guiden lagde punkter på forskjellig måte.

- ☐ Utvid punktene dine med en z-verdi. Hva skjer når du kopilerer? Klarer du tyde feilmeldingen?
- ☐ Lag en ny type: `Point3D` som også har Z-verdi, og spesifiser at punktene dine skal være av denne typen:

```
myPoint : Point3D
-- ...
```

Dette får vi bruk for!

Steg 4: Datastrukturer i datastrukturer

I steg 3 bygget vi opp datastrukturen `Point` fra to tall av typen `Float`.

Nå skal vi bruke vår egen type, `Point`, til å bygge opp ett kvadrat.

Desimaltall

Obs! Her kommer det matte. Viktig for oss nå:

Kommatall i Elm har typen `Float`.

Dette har en forklaring:

Desimaltall i Elm har typen `Float`. Float er kort for *Floating point number*, som på norsk er *flyttall*. Disse kalles flyttall fordi de har *flytende presisjon*. Det betyr at vi kan ha et fast antall *siffer* med nøyaktighet. Vi kan også lage veldig store tall, som $1000 * 1000 * 1000 * 1000 * 10000$

✓ Sjekkliste

☐ Hva må vi vite om et kvadrat for at vi skal kunne tegne det?

☐ Lag typen kvadrat: `type alias Square = -- ...`

Nå skal vi tegne kvadratet!

```
viewSquare square = -- ...
```

☐ Lag funksjonen `viewSquare`. Bruk `rect` fra SVG som du har brukt tidligere.

Obs! Når vi tegner kvadrater må vi bruke en farge. En måte å løse det på er å ha en `color : String`-attributt på `Square`.

Her er hvordan jeg bruker min `viewSquare`:

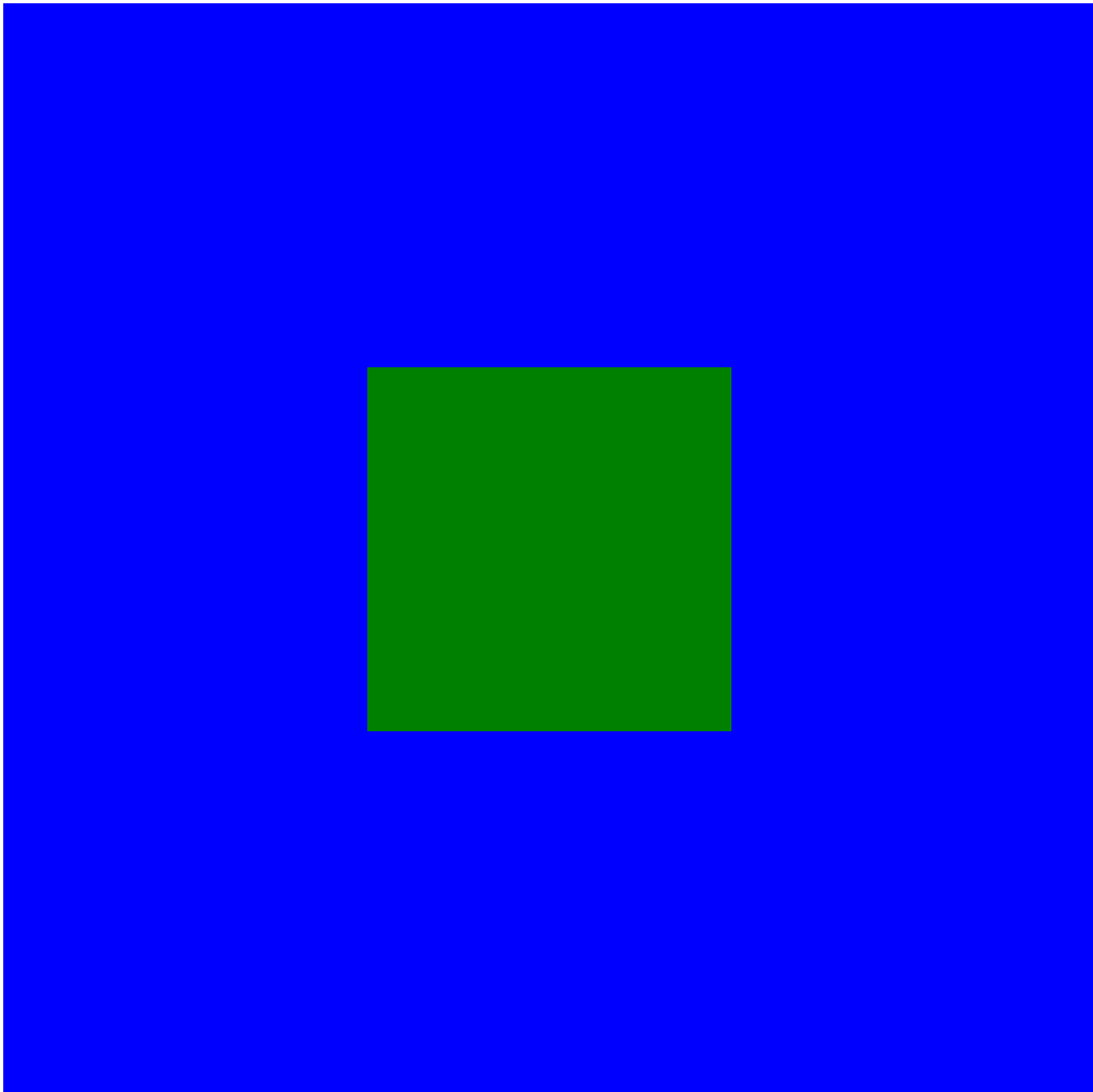
```
start =
  { corner = { x = 0.0
               , y = 0.0
             }
    , width = 27.0
    , color = "blue"
  }

center =
  { corner = { x = 9.0
               , y = 9.0
             }
    , width = 9.0
    , color = "green"
  }

main =
  div []
    [ h1 [] [ text (toString start)
                  , text (toString center)
                ]
      , svg
        [ width "500", height "500", viewBox "0 0 27 27" ]
        [ viewSquare start
          , viewSquare center
        ]
    ]
```

Dette blir seende slik ut på min PC:

```
{ corner = { x = 0, y = 0 }, width = 27, color = "blue" } { corner = { x = 9, y = 9 }, width = 9, color = "green" }
```



Nå har vi **startregelen** i boks! Den er kvadratet **start** !

Steg 5: Senterkvadrat og **let**

Vi kan sette binde navn med **let**. Her binder vi **age** til alderen vi regner ut:

```
describeAge yearNow yearBorn =  
  let age = yearNow - yearBorn  
  in "The age is " ++ (toString age)
```

Vi kan binde flere navn som kan være avhengig av hverandre:

```
describeHalfAge yearNow yearBorn =  
  let age = yearNow - yearBorn  
    halfAge / 2  
  in "Half the age is " ++ (toString halfAge)
```

✓ Sjekkliste

- ☐ Lag funksjonen **describeDoubleAge**. Hva skal denne gjøre?
- ☐ Les overskriften **Let expressions** i [syntaxguiden](#). Her er det noen eksempler. Prøv selv!

Nå skal vi tilbake til fraktalene våre, vi skal lage den grønne firkanten i sentrum. Denne gangen med kode!

- ☐ Lag funksjonen **centerSquare**. Denne skal ta inn et kvadrat og returnere kvadratet i sentrum av det forrige. Hvilken farge skal det ha?

Her er en start:

```
centerSquare : Square -> Square  
centerSquare old =  
  let x =
```

Du skal kunne bruke den slik:

```
> centerSquare  
<function> : Utils.Square -> Utils.Square  
> centerSquare start  
{ color = "blue", width = 9, corner = { x = 9, y = 9 } } : Utils.Square
```

... hva må x-verdien være om det nye kvadratet skal være i sentrum av det forrige?

```
<svg width="500" height="500" viewBox="0 0 27 27"><rect x="0" y="0" width="27" height="27" fill="blue"></rect><rect x="9" y="9" width="9" height="9" fill="green"></rect></svg>
```

Steg 6: Funksjoner fra `List` og `String` `{.activity}`

`List.map` kjører en funksjon på hvert element i en liste. Eksempel:

```
add1 x = x + 1 : number -> number
List.map add1 [10, 20, 30] [11,21,31] : List number

times2 x = x * 2 : number -> number
List.map times2 [10, 20, 30] [20,40,60] : List number
```

Sjekkliste `{.check}`

- Les avnsitet om `List.map` i [dokumentasjonen til `List`](<http://package.elm-lang.org/packages/elm-lang/core/latest/List#map>).
- Bruk `List.map` til å lage listen `["1","2","3","4"]`

`List.range` kan lage en liste med tall. Eksempel:

```
List.map toString (List.range 1 4) ["1","2","3","4"] : List String
List.range 5 10 [5,6,7,8,9,10] : List Int
List.range 0 3 [0,1,2,3] : List Int
```

- Les avsnittet om `List.range` i [dokumentasjonen til `List`](<http://package.elm-lang.org/packages/elm-lang/core/latest/List#range>)
- Bruk `List.map` og `List.range` til å lage denne store listen:

```
```elm
["0","1","2","3","4","5","6","7","8","9","10","11","12","13","14",
,"15","16","17","18","19","20","21","22","23","24","25","26","27",
,"28","29","30"]
```

Vi innfører enda en nyttig funksjon: `String.join`. Denne bygger opp tekst fra en liste.

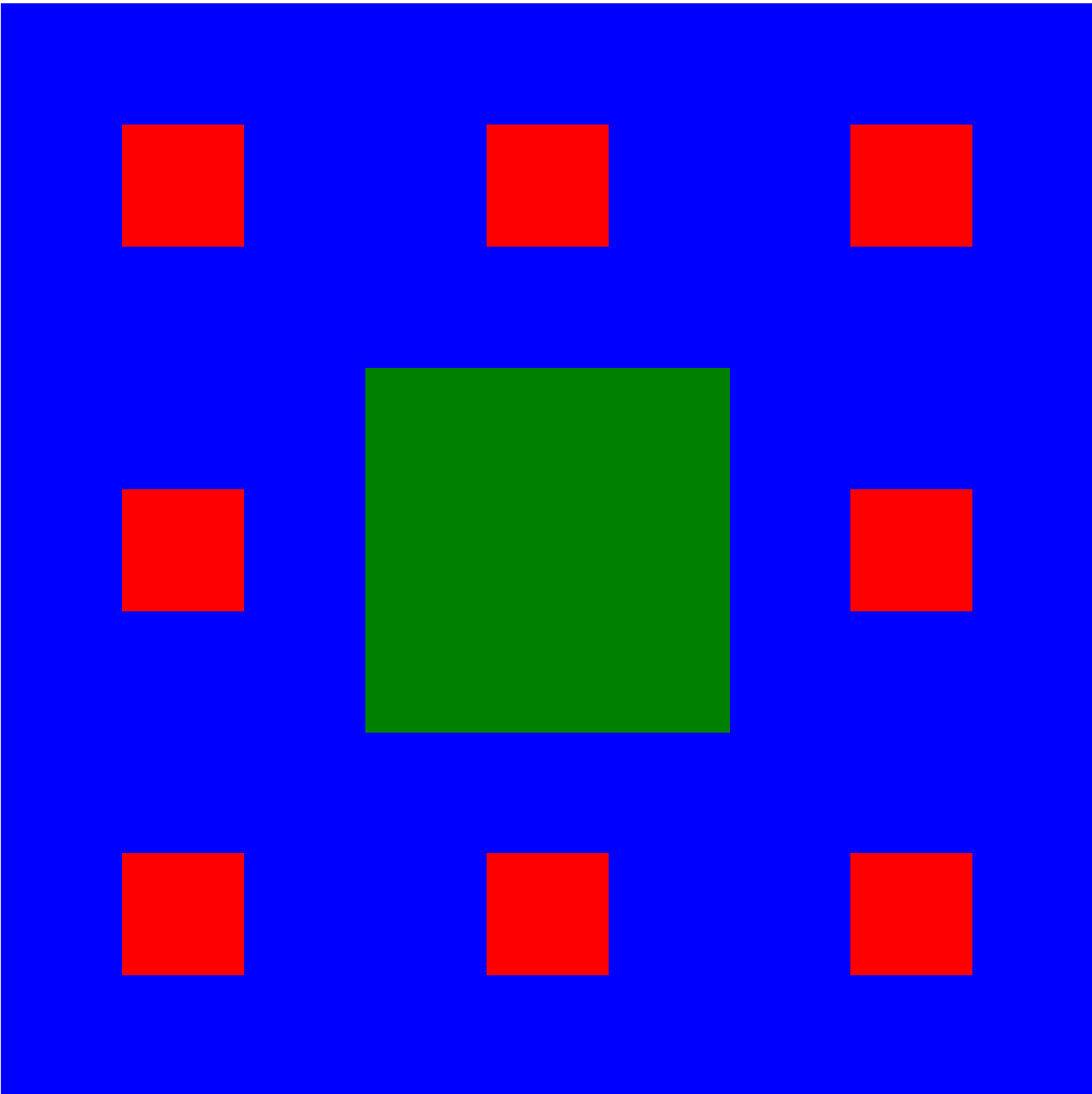
- ☐ Les dokumentasjonen til `String.join` i [dokumentasjonen til `String`]
- ☐ Lag funksjonen `sayTo`. Den skal kunne brukes slik:

```
> sayTo 10
"1 og 2 og 3 og 4 og 5 og 6 og 7 og 8 og 9 og 10" : String
> sayTo 3
"1 og 2 og 3" : String
```

Bra! Gi deg selv en klapp på skulderen.

## Steg 7: Kvadrater langs kanten

Nå skal vi finne kvadratene langs kanten. Hvor mange blir det? Tell de røde:



**Obs!** Denne er en utfordring. Ta deg god tid.

## ✓ Sjekkliste

☐ Lag funksjonen `borderSquares` . Denne skal vi kunne bruke slik:

```
> start
{ color = "green", width = 729, corner = { x = 0, y = 0 } } : Utils.Square
> borderSquares start
[{ color = "blue", width = 243, corner = { x = 0, y = 0 } }
,{ color = "blue", width = 243, corner = { x = 243, y = 0 } }
,{ color = "blue", width = 243, corner = { x = 486, y = 0 } }
,{ color = "blue", width = 243, corner = { x = 0, y = 243 } }
,{ color = "blue", width = 243, corner = { x = 486, y = 243 } }
,{ color = "blue", width = 243, corner = { x = 0, y = 486 } }
,{ color = "blue", width = 243, corner = { x = 243, y = 486 } }
,{ color = "blue", width = 243, corner = { x = 486, y = 486 } }]
: List Utils.Square
```

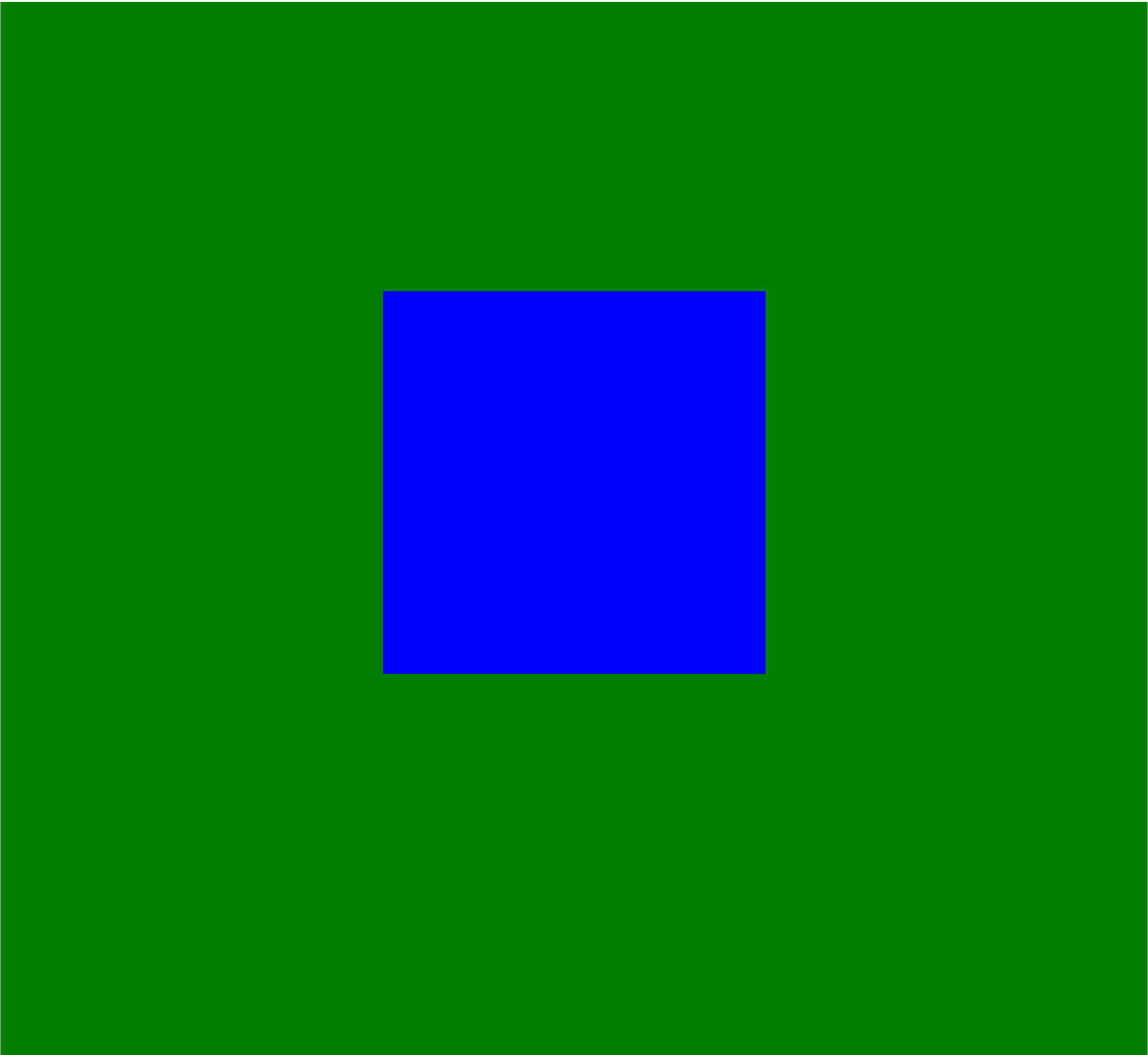
## Steg XX: Tegn fraktalen

**OBS!** Teodor ble ikke ferdig med oppgaven! Her er hva vi vil ende opp med:

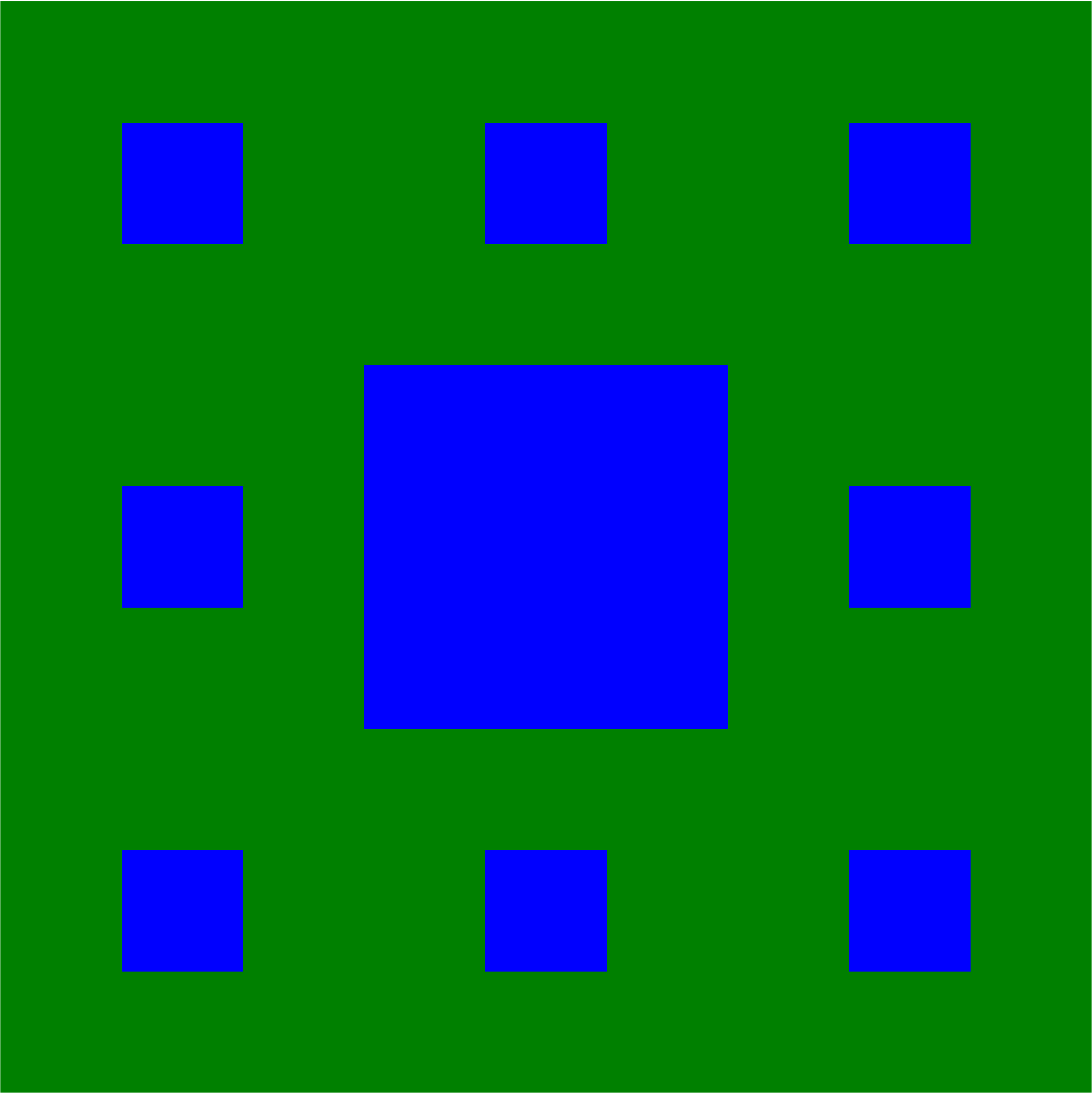
**Nivå 1:**



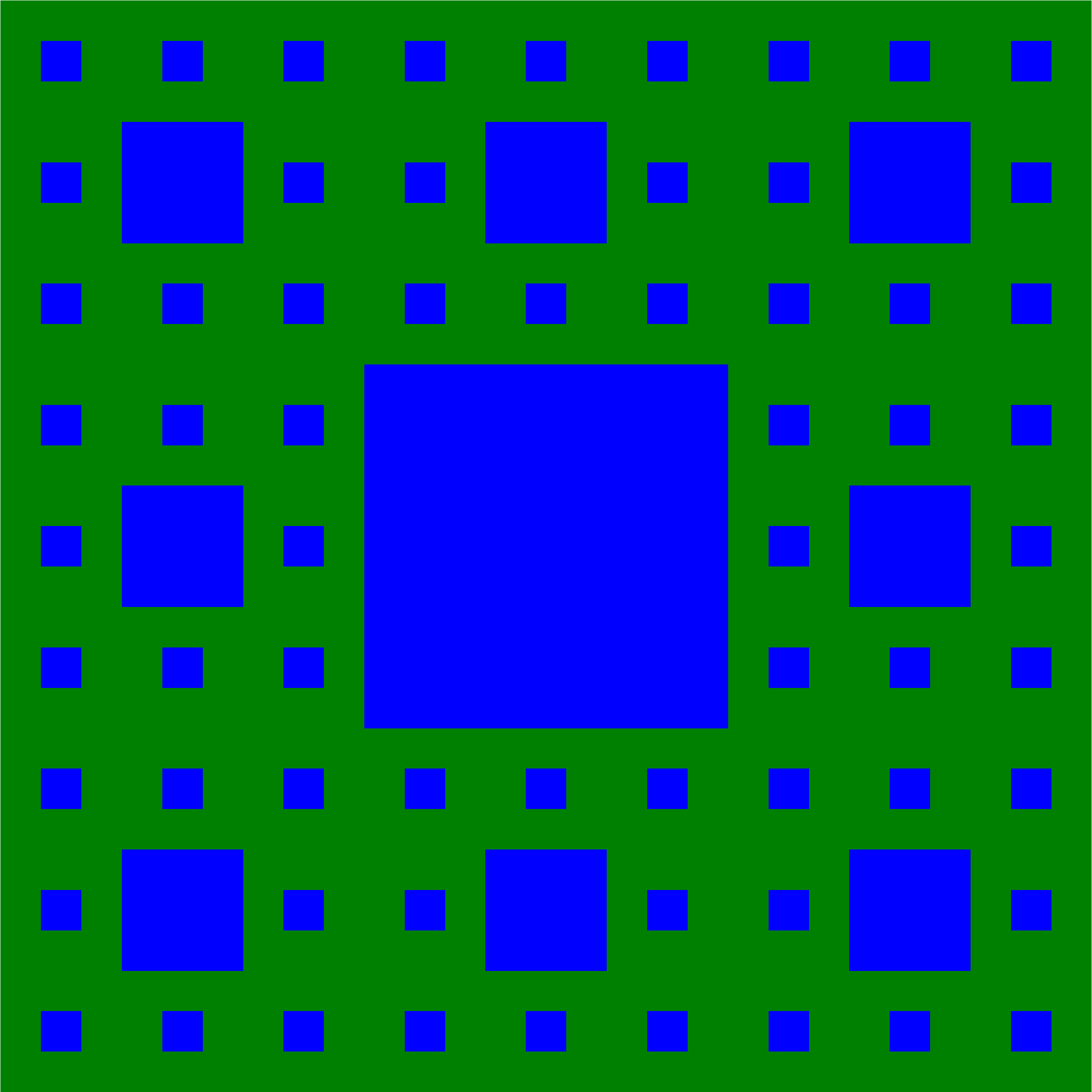




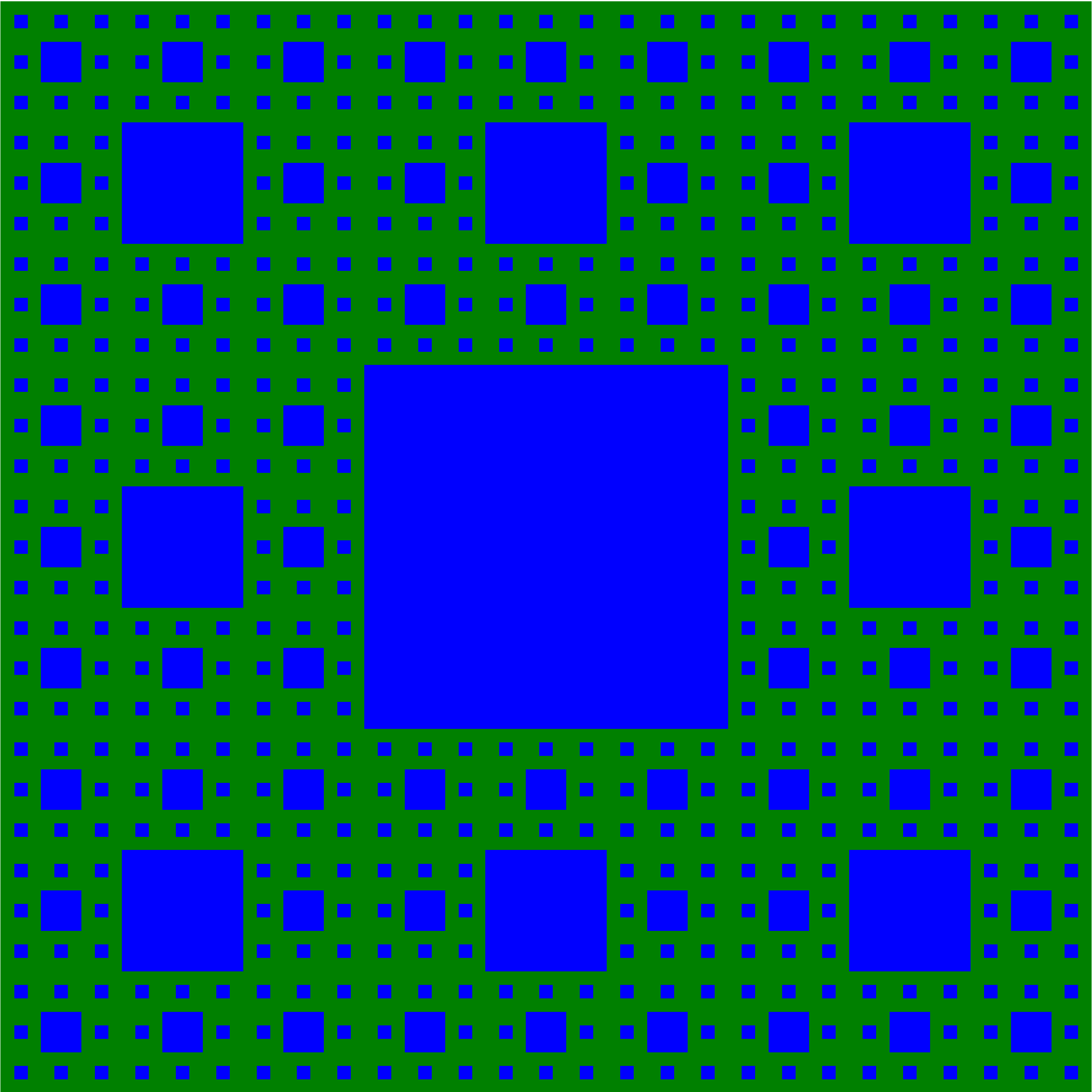
Nivå 2:



Nivå 3:



Nivå 4:



... nå tar det så lang tid på min PC at jeg ikke gidder lenger inn!

Prøv videre selv, så kommer jeg tilbake til neste uke.

**Lisens:** [CC BY-SA 4.0](#) **Forfatter:** Teodor Heggelund