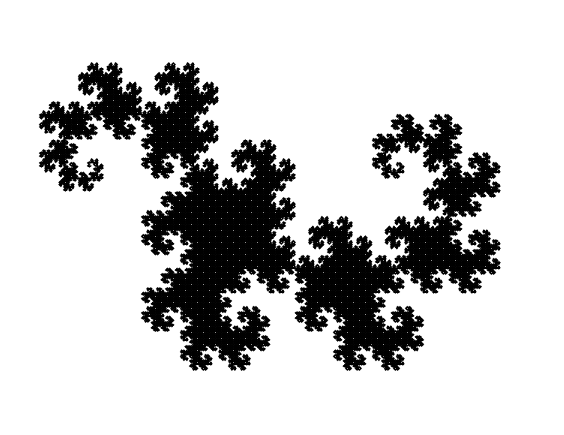
Grafische Darstellung von Fraktalen

Portfolioarbeit

Von Jonah Sebright

Klasse 6A



EF-Lehrer: PD Dr. Victor Yakhontov

Gymnasium Kirschgarten Basel 2021

## Inhaltsverzeichnis

[1. Einleitung 2](#_Toc71983956)

[1.1 Was sind Fraktale? 2](#_Toc71983957)

[1.2 Ziele des Projekts 2](#_Toc71983958)

[1.3 Vorschau Resultate 2](#_Toc71983959)

[2. Hauptteil (Material und Methoden) 3](#_Toc71983960)

[2.1 Programmiersprache 3](#_Toc71983961)

[2.2 Bibliotheken 3](#_Toc71983962)

[2.2.1 StdDraw 3](#_Toc71983963)

[2.2.2 Flanagan 3](#_Toc71983964)

[2.3 Reduktion des Problems 4](#_Toc71983965)

[2.4 Entwicklung der Module 5](#_Toc71983966)

[2.5 Entwicklung der Algorithmen 5](#_Toc71983967)

[2.5.1 Schneeflocke 5](#_Toc71983968)

[2.5.2 Pfeilspitze 6](#_Toc71983969)

[2.5.3 Drachenkurve 6](#_Toc71983970)

[2.6 Testen 6](#_Toc71983971)

[3. Resultate 6](#_Toc71983972)

[3.1 Grafisch dargestellte Fraktale 6](#_Toc71983973)

[3.2 Effizienz 7](#_Toc71983974)

[3.3 Features des Main-Programms? Oder zu einleitung?? 7](#_Toc71983975)

[4. Diskussion 7](#_Toc71983976)

[Flächedrachenkurve 7](#_Toc71983977)

[4.1 Optimierungsmöglichkeiten 7](#_Toc71983978)

[4.2 Rückblick auf das Problem 8](#_Toc71983979)

[5. Nachwort? 8](#_Toc71983980)

[6 Quellenverzeichnis 8](#_Toc71983981)

[7. Abbildungsverzeichnis 8](#_Toc71983982)

[8. Tabellenverzeichnis 8](#_Toc71983983)

[9. Quellcodeverzeichnis 8](#_Toc71983984)

[10. Anhang 8](#_Toc71983985)

[10.1 Weitere Diagramme 8](#_Toc71983986)

[10.2 Gesamter Quellcode 8](#_Toc71983987)

# 1. Einleitung

## 1.1 Was sind Fraktale?

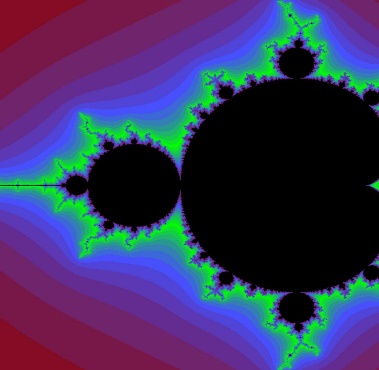


Abb. 1 Mandelbrot Fraktal ([Vater der Fraktale: Mandelbrot bewies die Schönheit der Mathematik - WELT](https://www.welt.de/wissenschaft/article10361079/Mandelbrot-bewies-die-Schoenheit-der-Mathematik.html), 2010)

Fraktale sind geometrische Gebilde, die sich in nicht ganzzahligen („fraktalen“) Dimensionen befinden. Benoît Mandelbrot zeigte erstmals, dass Fraktale überall in der Natur existieren. Fraktale werden beispielsweise verwendet, um Längen von Küsten zu berechnen (vgl. [Fraktale - Lexikon der Physik (spektrum.de)](https://www.spektrum.de/lexikon/physik/fraktale/5252), kein Datum).

## 1.2 Ziele des Projekts

Das Hauptziel dieses Projekts ist, mit den gelernten Fähigkeiten und dem erworbenen Wissen ein konkretes Informatik-Problem zu lösen. Das in diesem Projekt untersuchte Problem soll bestimmte Fraktale, wie die „Schneeflocke“, die „Pfeilspitze“ und die „Drachenkurve“, visuell darstellen. Das Vorgehen beim Lösen des Problems wird ebenfalls in dieser Portfolioarbeit dargestellt.

Zu diesem Problem soll ein Programm erstellt werden, das dem Benutzer eine Annäherung des ausgewählten Fraktals bis zu einer gewünschten Stufe grafisch wiedergibt. Zur Zeichnung der Kurve soll die Zeichnungs-Bibliothek „StdDraw“ verwendet werden. Sobald die Darstellung vollendet ist, wird der Benutzer gefragt, ob er die Zeichnung als PDF-Datei speichern möchte. Dabei kann der Benutzer den Dateinamen selbst eingeben.

Für die grafische Darstellung der Fraktale müssen die Algorithmen der einzelnen Fraktale geschrieben werden, die alle das Basisprinzip der Rekursion implementieren. Zudem verwenden alle Algorithmen die Hilfs-Klasse *Schildkroete* INSERT\_LINK\_TO implemtierung modules.

Ausserdem werden in diesem Projekt die Resultate des Programms analysiert und mit mathematischen Erkenntnissen verglichen. Bei der Drachenkurve wird in der Diskussion bewiesen, dass die unendlich lange Kurve eine endliche Fläche bedeckt.

Struktur der Arbeit

## 1.3 Vorschau Resultate

Die Zeichnungen des Programms sind zum Staunen schön! Das Haupt-Programm läuft ohne Fehler und beinhaltet alle oben beschrieben Funktionen und erfüllt somit dessen Ziele. Die einzelnen Algorithmen der Fraktale funktionieren ebenfalls einwandfrei bei beliebiger Stufe. Hier sind ein paar grafische Resultate der verschiedenen Fraktale:

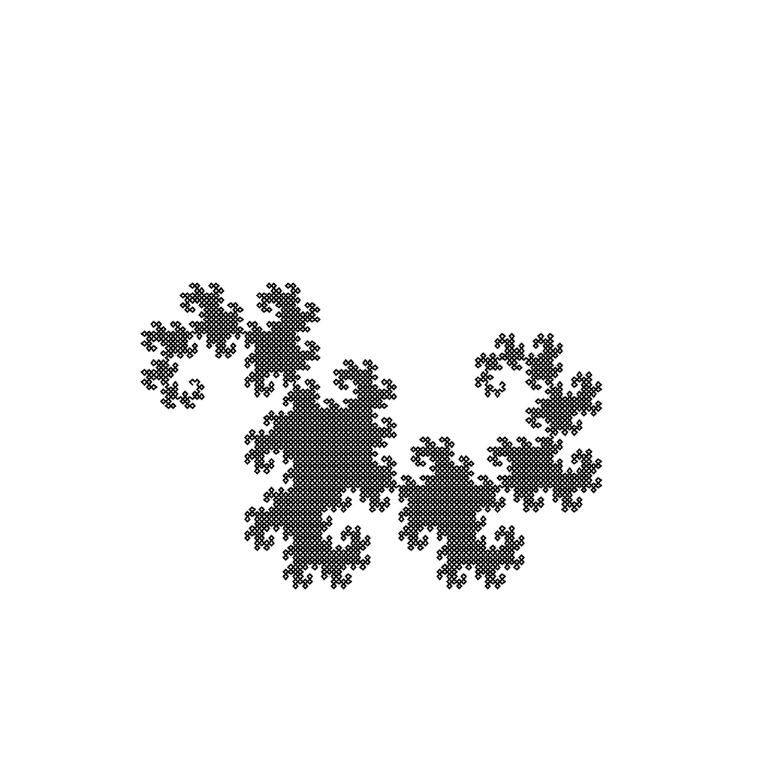


Abb. 2 Drachenkurve bei Stufe 13

Abb. 3 Schneeflocke bei Stufe 7

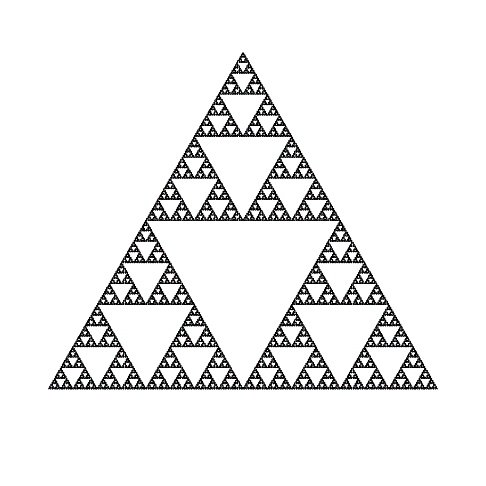
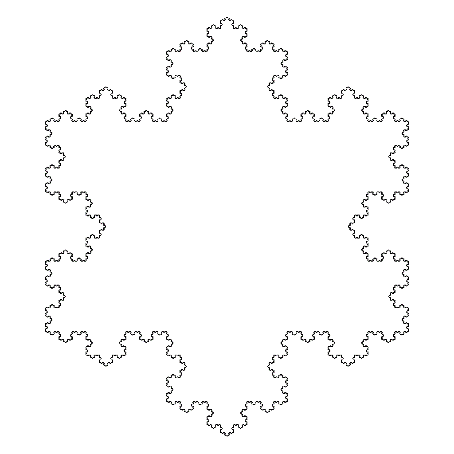


Abb. 4 Pfeilspitze bei Stufe 9

Was und wie gemacht und was wird vorgsestellt, Darstellung

Das Programm besteht aus den Modulen *Main, Schildkroete*, *Fraktal, Input* und den Bibliotheken *StdDraw* und *Flanagan* (siehe Abb. 5 Modularisierung des Programms).

Insgesamt wurden 13 Klassen erstellt mit ca. 500 Linien Quellcode. Eine Gesamtübersicht der Klassen und ihren Beziehungen ist im Anhang in INSERT\_REFERENCE ersichtlich.

# 2. Hauptteil (Material und Methoden)

Eine Java-Applikation ist sehr komplex und kann durch falsche Entscheidungen schnell zu Problemen führen. Geschicktes Vorgehen ist daher sehr wichtig und die Grundstrukturen müssen stimmen, bevor Details implementiert werden können. Besonders mit zunehmendem Umfang des Programms ist die Architektur umso wichtiger.

Begonnen wird mit der konkreten Definition des Programms, was schon in der Einleitung gemacht wurde. Danach findet die Reduktion des Problems statt, wobei die Module und deren Beziehungen und der Grobalgorithmus skizziert werden. Anschliessend können die einzelnen Module und Algorithmen konkret implementiert werden. Möglicherweise werden dafür Bibliotheken verwendet (siehe 2.2.1 StdDraw). Erst in diesem Schritt wird der Quellcode verfasst.

Damit das Programm stabil und sicher ist, wurde es mit den erwarteten sowie auch unerwarteten Eingabewerten des Benutzers getestet. Dies ist wichtig, da Benutzer nicht immer die erwarteten Werte eingeben und Fehler des Programms zu vermeiden sind. Zudem muss auch die Effizienz getestet werden, sodass ein langsames Programm bemerkt und möglicherweise optimiert werden kann.

Für die Verwendung des Programms muss zunächst eine Verbindung zwischen dem Benutzer und dem Programm hergestellt werden. Dies wird durch ein GUI (graphical user interface) gemacht, was die Interaktion erlaubt. In diesem Programm müssen lediglich ein paar Fenster zum Einlesen und zur Resultat-Ausgabe geöffnet werden. Die Bibliothek „Flanagan“ (siehe 2.2.2 Flanagan) erfüllt diese Anforderungen.

## 2.1 Programmiersprache

Wie für die Arbeit vorgegeben wurde das Programm ausschliesslich mit Java programmiert. Java ist eine objekt-orientierte Programmiersprache von Oracle und bietet so viele Vorteile, wie unter anderem Polymorphie (vgl. Ullenboom 2012, p .50).

## 2.2 Bibliotheken

Was sind bibliotheken?

### 2.2.1 StdDraw

Für das Programm wurde eine Zeichnungsbibliothek namens „StdDraw“ vom *Computer Science* *Department* der Universität Princeton verwendet (vgl. [Computer Science Department at Princeton University](https://www.cs.princeton.edu/), 2021). Die Bibliothek bietet sowohl eine Zeichenfläche als auch Methoden an, mit denen die Zeichenfläche manipuliert werden kann. Die wichtigsten Methoden für dieses Projekt sind *line, setPenColor, setPenRadius, clear*. Eine weitere nützliche Funktion ist *save*, mit der die Zeichenfläche als Bild-Datei gespeichert werden kann.

### 2.2.2 Flanagan

Für das Einlesen der gewünschten Kurve, der Stufe und des Dateinamens des zu speichernden Bildes wurde die Flanagan-Bibliothek von M. Thomas ([Michael Thomas Flanagan's Java Scientific Library: Input through a dialog box (ucl.ac.uk)](https://www.ee.ucl.ac.uk/~mflanaga/java/Db.html), 2010) implementiert. Mit dieser können einfache Eingabe-Fenster geöffnet werden. Die wichtigsten Methoden für dieses Projekt sind *optionBox*, *readInt*, *yesNo* und *readLine*, um die notwendige Information von dem Benutzer zu erhalten.

## 2.3 Reduktion des Problems

Komplexe Systeme können in kleinere Aufgaben geteilt und so einfacher gelöst werden. Dieses Aufteilen heisst **Modularisierung**. Jede einzelne Aufgabe kann für sich gelöst werden.

Dieses Programm habe ich folgendermassen modularisiert:

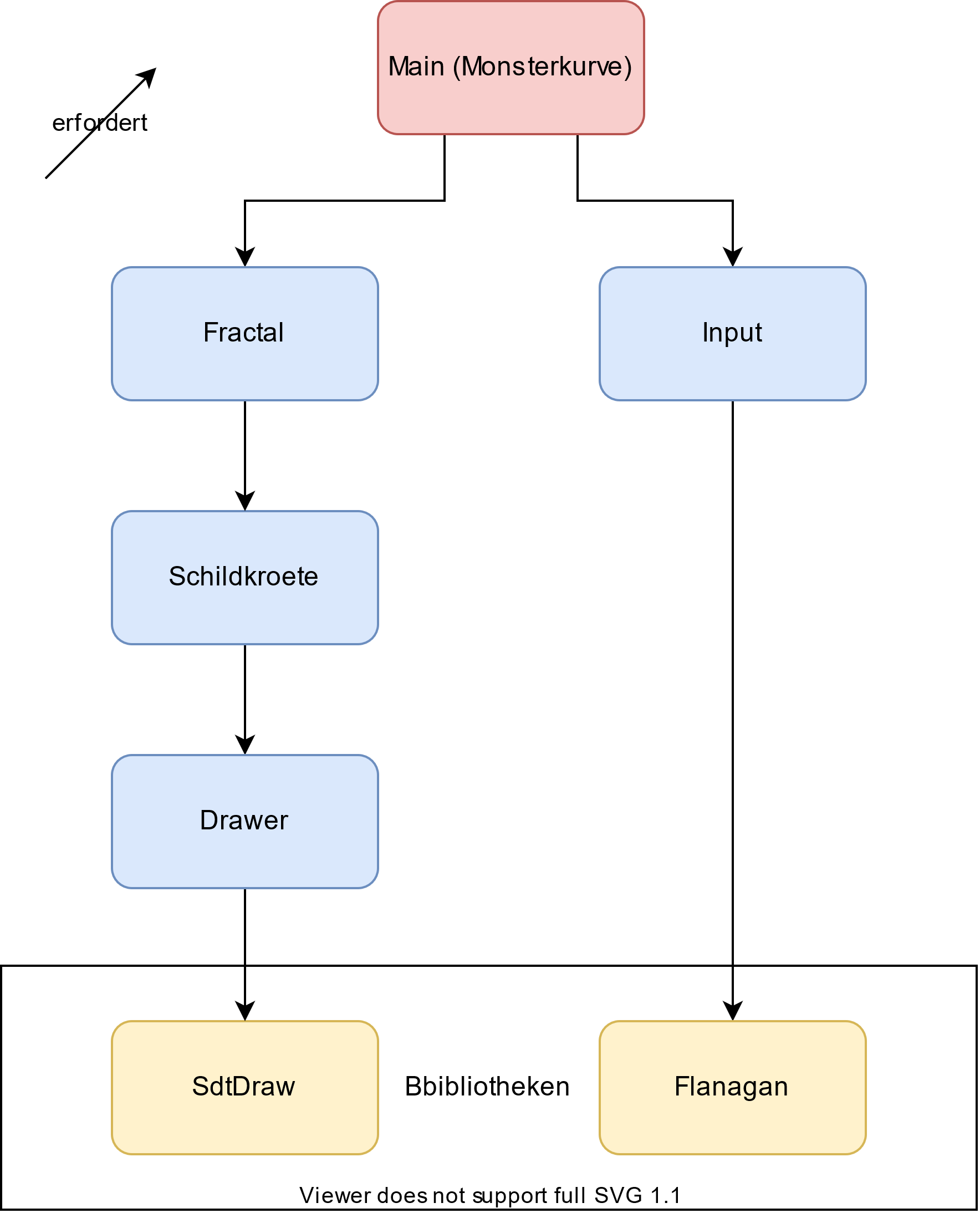


Abb. 5 Modularisierung des Programms

*Main* erfordert das Hilfsmodul *Input*, welches auf die Bibliothek *Flanagan* angewiesen ist. Zudem erfordert *Main* die *Fraktale* (Schneeflocke, Pfeilspitze, Drachenkurve), die das Modul *Schildkroete* verwenden. *Schildkroete* erfordert das Modul *Drawer*, welches unter anderem die Methode *drawLine(Point from, Point to)* bietet. Ausserdem verwendet *Drawer* die Bibliothek *StdDraw*.

Nachdem die Module entworfen wurden, können sie und ihr Zusammenspiel implementiert werden. Dabei muss zunächst ein **Grobalgorithmus** entworfen werden. Der Grobalgorithmus des Main-Programmes ist in Abb. 6 ersichtlich.

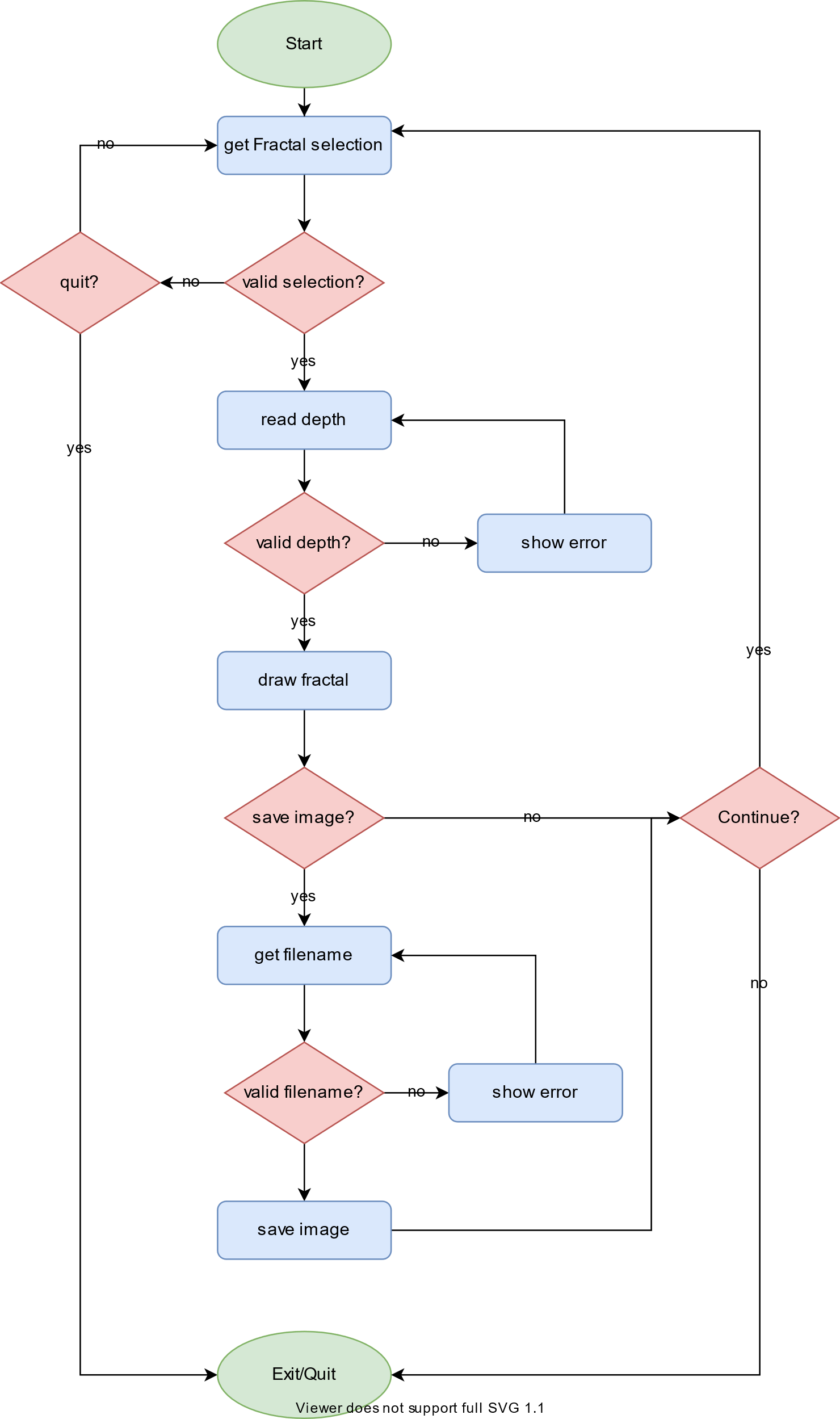


Abb. 6 Grobalgorithmus des Main Programms

Bei allen drei Fraktalen beruht der Grobalgorithmus auf dem allgemeinen Prinzip der Rekursion. Dies bedeutet, dass eine Methode sich selbst immer wieder aufruft, bis eine End-Bedingung erfüllt ist (siehe Abb. 7 Ablaufsprinzip der Rekursion).

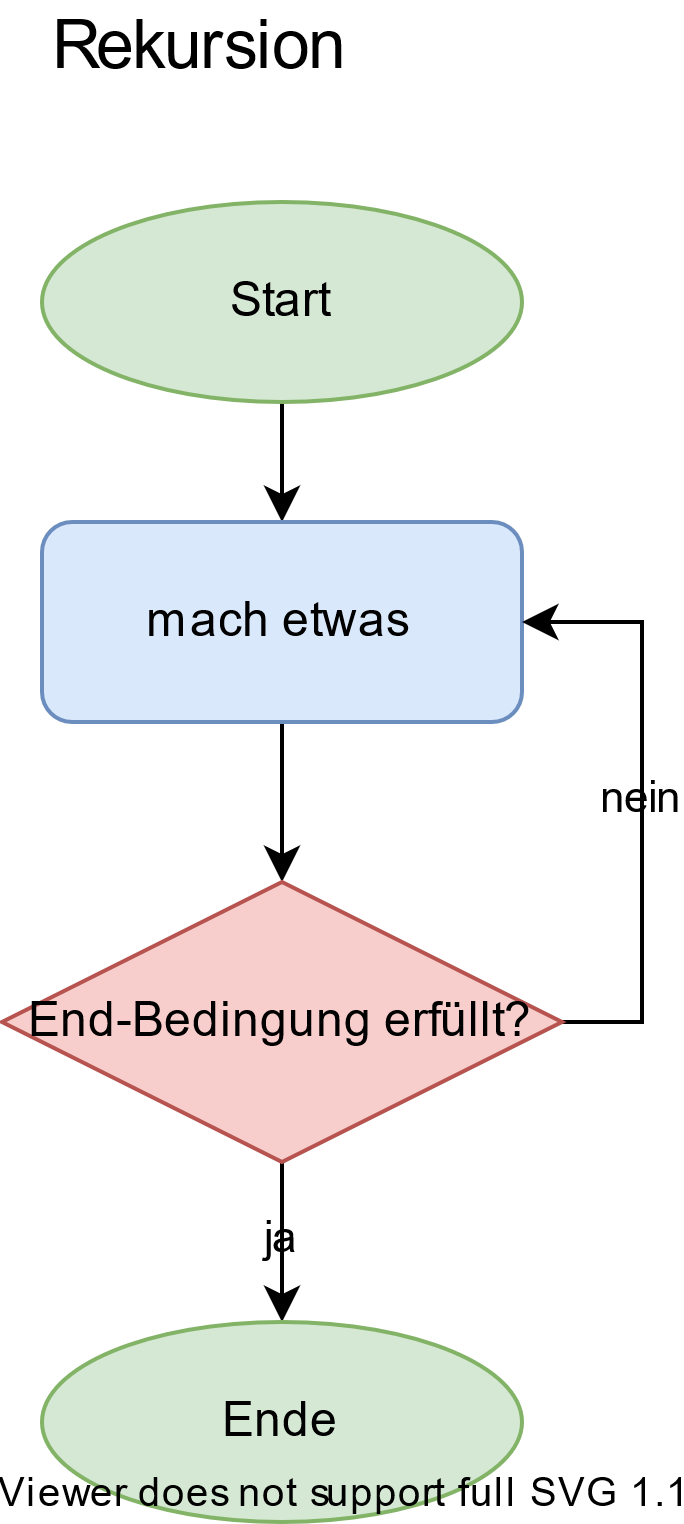


Abb. 7 Ablaufsprinzip der Rekursion

## 2.4 Entwicklung der Module

Ablaufdiagramme, Beschreibung variablen

## 2.5 Entwicklung der Algorithmen

Wie erwähnt, wird zu Konstruktion aller drei Kurven die Rekursion verwendet. Bei den drei Algorithmen sind die Parameter *depth (int)* und *length (double)* gegeben. Jene manipulieren die Richtung der *Schildkroete* und rufen ihre eigene Methode wieder mit veränderten Parametern auf. Die Stufe (*depth*) wird um eins verkleinert und die Strecke (*length*) durch einen konstanten Faktor dividiert. Wenn die Stufe den Wert null erreicht, wird die gegebene Strecke gezeichnet und die Methode ruft sich nicht mehr auf. Die End-Bedingung wurde also erfüllt und die Methode beendet.

Das Grundprinzip ist, dass der *Initiator* (anfängliche Strecke) bei jeder Iteration durch den *Generator* ersetzt wird. Der Generator ersetzt also eine gerade Strecke durch eine komplexere Kurve, die eine grössere Anzahl, dafür aber kürzere gerade Strecken aufweist. Diese Strecken werden in der nächsten Stufe wiederum durch den *Generator* ersetzt. So wird dies für jede Stufe wiederholt, bis im Unendlichen die richtige Kurve erreicht wurde.

### 2.5.1 Schneeflocke

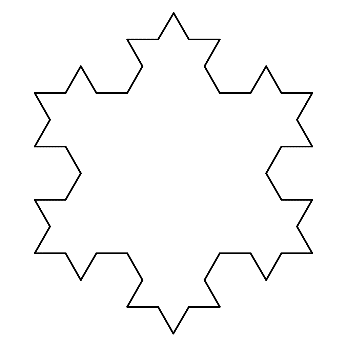


Abb. 8 Schneeflocke 2

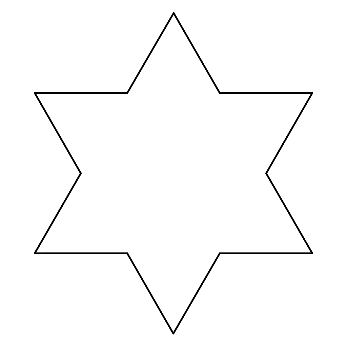


Abb. 9 Schneeflocke 1

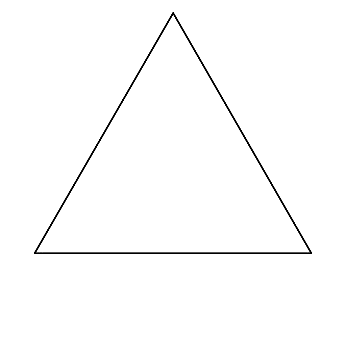


Abb. 10 Schneeflocke 0

Die Schneeflockenkurve ist eine der simpelsten und schönsten Kurven, die rekursiv definierbar sind. Ihr *Initiator* ist in diesem Fall ein gleichseitiges Dreieck. Die nächste Stufe entsteht dadurch, dass jede gerade Strecke der vorherigen Stufe durch drei geteilt und der mittlere Drittel durch zwei Seiten eines gleichseitigen Dreiecks ersetzt wird (siehe Abb. 10, Abb. 9, Abb. 8).

INSERT\_CODE

### 2.5.2 Pfeilspitze

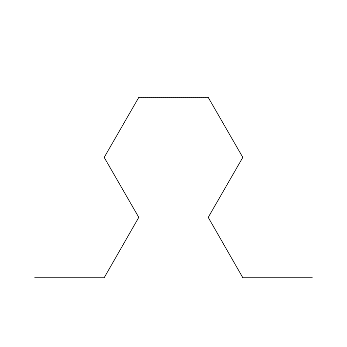


Abb. 11 Pfeilspitze 2

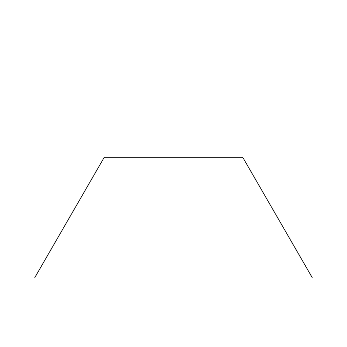


Abb. 12 Pfeilspitze 1



Abb. 13 Pfeilspitze 0

Die Pfeilspitzenkurve (auch Sierpinski-Dreieck genannt) hat eine gerade Strecke als *Initiator*. Ihr *Generator* ist eine dachförmige Figur mit 60 Grad Winkeln, wobei alle Seiten die gleiche Länge haben. Man könnte sagen, es ist die obere Hälfte eines Sechsecks. Allerdings wechselt die Richtung des *Generators* ab, um den gewünschten Effekt zu erhalten (siehe Abb. 13, Abb. 12, Abb. 11). Deshalb wird der zusätzlicher Parameter *richtung (boolean)* für die Funktion eingebaut, der von Stufe zu Stufe seinen Wert umkehrt.

INSERT\_CODE

### 2.5.3 Drachenkurve

## 2.6 Testen

Testen ist für jede Applikation essentiell, um geringe als auch signifikante Fehler zu vermeiden. Dabei ist es wichtig, nicht nur die optimalen Eingabe-Werte des Benutzers zu erwarten, sondern auch die ungewöhnlichsten Werte. Diese App wurde mit diversen Werten getestet und so gebaut, dass sie dem Benutzer im Falle falscher Eingaben sowohl konstruktive Fehlermeldungen anzeigt als auch Korrekturmöglichkeiten anbietet.

Ebenfalls wichtig ist die Effizienz der Algorithmen. Diese wurde gemessen und die Resultate werden in 3.2 Effizienz aufgezeigt und analysiert. Dafür wurden die Laufzeit und die Anzahl der Bewegungen der *Schildkroete* in Betracht gezogen.

# 3. Resultate

## 3.1 Grafisch dargestellte Fraktale

Die grafischen Resultate der Fraktale bei verschieden Stufen werden im folgenden Abschnitt präsentiert:

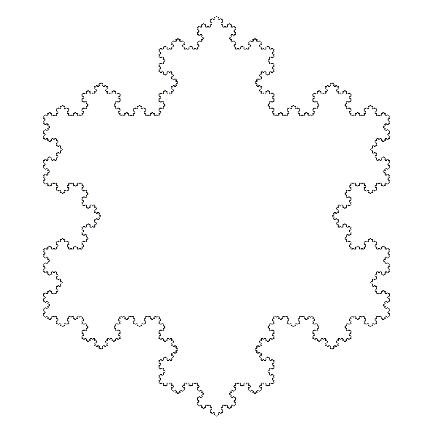


Abb. 14 Schneeflocke 6

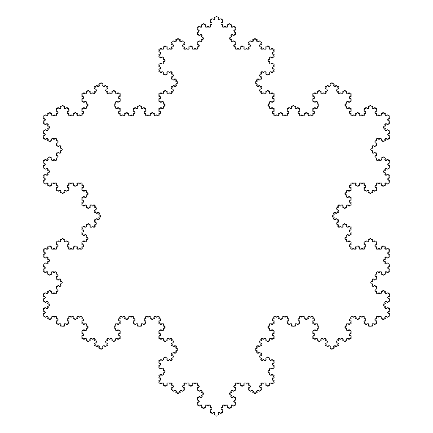


Abb. 15 Schneeflocke 8

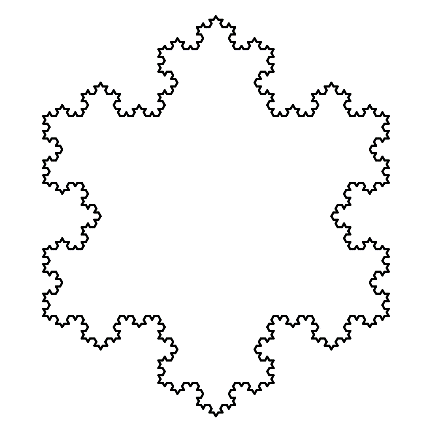


Abb. 16 Schneeflocke 4

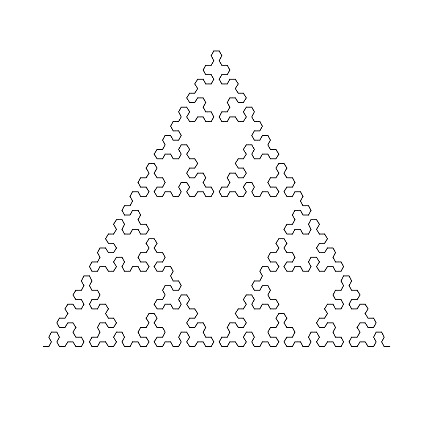
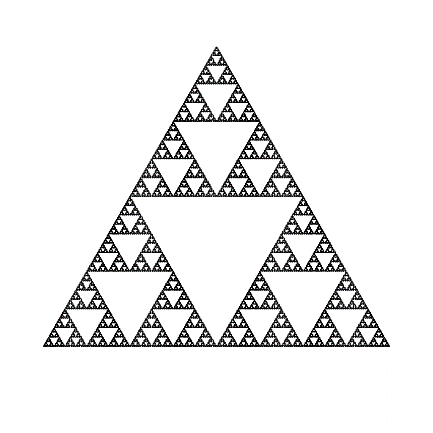


Abb. 17 Pfeilspitze 6



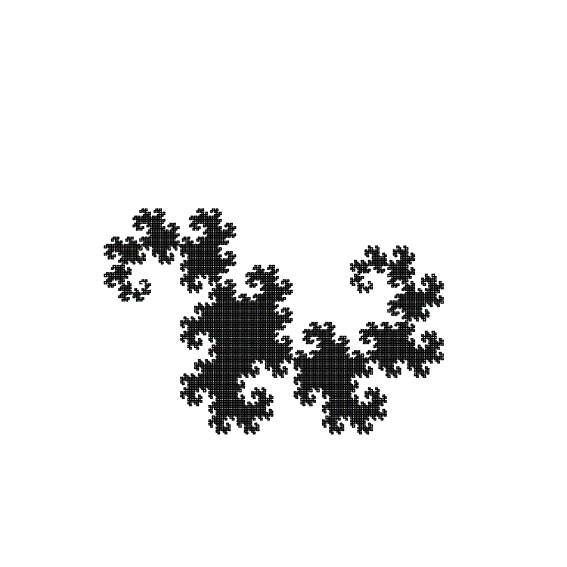


Abb. 18 Drachenkurve 14

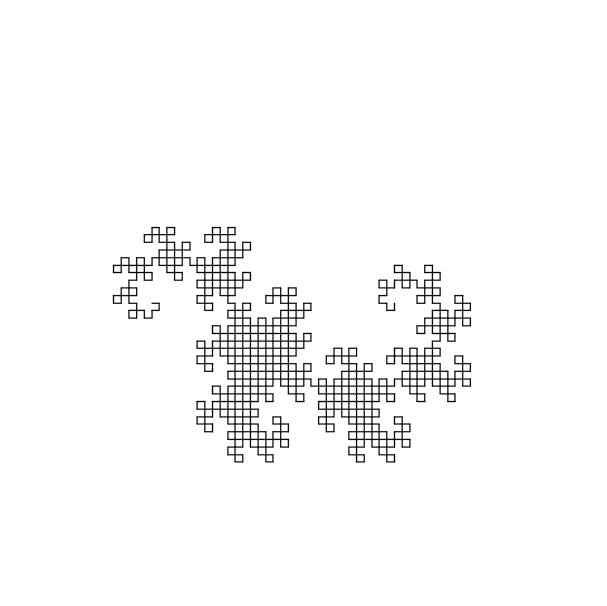


Abb. 19 Drachenkurve 10

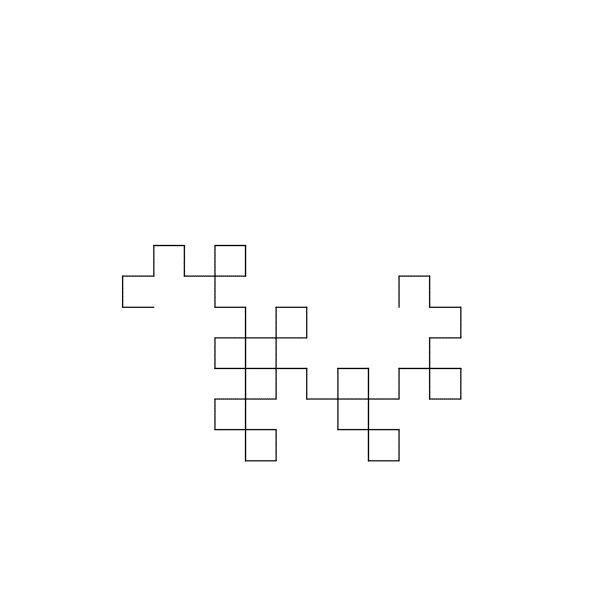


Abb. 20 Drachenkurve 6

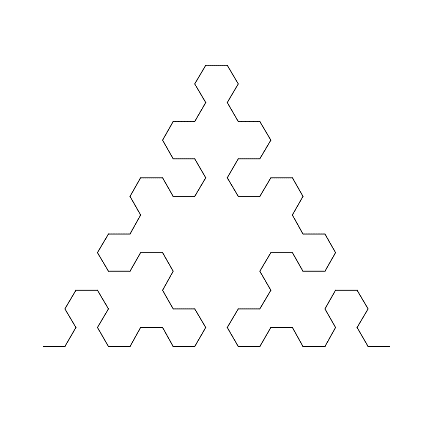


Abb. 21 Pfeilspitze 4

Abb. 22 Pfeilspitze 9

## 3.2 Effizienz

Bei vielen Applikationen spielt die Effizienz eine kleine Rolle. Ob das Programm 20 oder 50 Millisekunden beansprucht, ist für den Benutzer von geringer Bedeutung. Wenn es aber eine ganze Sekunde oder gar mehrere Minuten dauert, muss auf die Effizienz/Lauftzeit geachtet werden. Bei meinem Programm ist dies der Fall: Schon bei einer Stufe 5 bemerkt man eine deutlich erhöhte Laufzeit, was zwar in einer schönen Animation der Kurve resultiert aber nicht erwünscht ist. Bei einer Stufe von acht bei der Schneeflockenkurve werden sogar einige Minuten benötigt, um die Grafik zu vollenden.

INSERT\_TABLE , GRAPH

## 3.3 Features des Main-Programms? Oder zu einleitung??

# 4. Diskussion

## Flächedrachenkurve

blabla

## 4.1 Optimierungsmöglichkeiten

Die Resultate der Effizienz (siehe 3.2 Effizienz) zeigen eine drastische Erhöhung der Laufzeitdauer mit einer erhöhten Stufe. Genauer gesagt verhält sie sich exponentiell zur Stufe. Wenn *s* die Anzahl der Bewegungen der Schildkroete ist und *n* die Stufe, so ist bei der Schneeflocke , bei der Pfeilspitze und bei der Drachenkurve . Die Laufzeit ist proportional zu *s*.

Dieses exponentielle Verhalten lässt sich erklären: Bei der **Schneeflocke** beispielsweise besteht der Generator aus vier Seiten. Mit jeder Erhöhung der Stufe müssen also viermal mehr Strecken gezeichnet werden, deshalb die Formel . Bei der **Pfeilspitze** besteht der Generator aus drei Strecken, also müssen pro Stufe dreimal so viele Strecken gezeichnet werden, deshalb die Formel. Bei der **Drachenkurve** ist es gleich, nur ersetzt der Generator eine durch zwei Strecken.

Logisch gesehen ist es folglich unmöglich, die Effizienz zu steigern und eine geringere Laufzeit zu erhalten. Dies lässt sich leider nicht ändern.

## 4.2 Rückblick auf das Problem

Was für Schwierigkeiten?

Hätte besser angehen können?

# 5. Nachwort?

# 6 Quellenverzeichnis

# 7. Abbildungsverzeichnis

[Abb. 1 Mandelbrot Fraktal (Vater der Fraktale: Mandelbrot bewies die Schönheit der Mathematik - WELT, 2010) 2](file:///C:\Users\iwein\Documents\GitHub\EFI\Portfolio\Portfolio.docx#_Toc71968922)

[Abb. 2 Drachenkurve bei Stufe 13 2](file:///C:\Users\iwein\Documents\GitHub\EFI\Portfolio\Portfolio.docx#_Toc71968923)

[Abb. 3 Schneeflocke bei Stufe 7 2](file:///C:\Users\iwein\Documents\GitHub\EFI\Portfolio\Portfolio.docx#_Toc71968924)

[Abb. 4 Pfeilspitze bei Stufe 9 2](file:///C:\Users\iwein\Documents\GitHub\EFI\Portfolio\Portfolio.docx#_Toc71968925)

[Abb. 5 Modularisierung des Programms 4](file:///C:\Users\iwein\Documents\GitHub\EFI\Portfolio\Portfolio.docx#_Toc71968926)

[Abb. 6 Grobalgorithmus des Main Programms 4](file:///C:\Users\iwein\Documents\GitHub\EFI\Portfolio\Portfolio.docx#_Toc71968927)

[Abb. 7 Ablaufsprinzip der Rekursion 5](file:///C:\Users\iwein\Documents\GitHub\EFI\Portfolio\Portfolio.docx#_Toc71968928)

[Abb. 10 Schneeflocke 2 5](file:///C:\Users\iwein\Documents\GitHub\EFI\Portfolio\Portfolio.docx#_Toc71968929)

[Abb. 8 Schneeflocke 1 5](file:///C:\Users\iwein\Documents\GitHub\EFI\Portfolio\Portfolio.docx#_Toc71968930)

[Abb. 9 Schneeflocke 0 5](file:///C:\Users\iwein\Documents\GitHub\EFI\Portfolio\Portfolio.docx#_Toc71968931)

# 8. Tabellenverzeichnis

# 9. Quellcodeverzeichnis

# 10. Anhang

## 10.1 Weitere Diagramme

## 10.2 Gesamter Quellcode