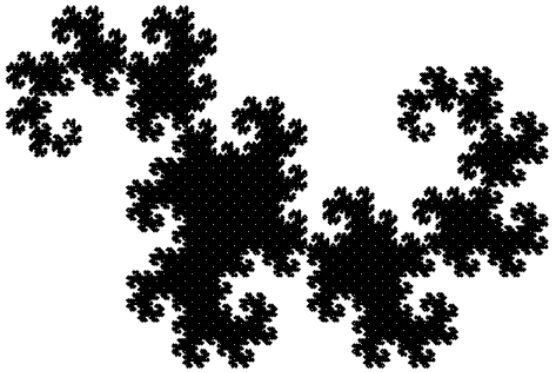
GRAFISCHE DARSTELLUNG VON FRAKTALEN

Portfolioarbeit

Von Jonah Sebright

Klasse 6A



Gymnasium Kirschgarten Basel 2021 EF-Lehrer: PD Dr. Victor Yakhontov

Inhaltsverzeichnis

1. Einleitung 2
1.1 Was sind Fraktale? 2
1.2 Ziele des Projekts 2
1.3 Vorschau Resultate 2
2. Hauptteil (Material und Methoden) 3
2.1 Programmiersprache 3
2.2 Bibliotheken 3
2.2.1 StdDraw 3
2.2.2 Flanagan 3
2.3 Reduktion des Problems 4
2.4 Entwicklung der Module 5
2.5 Entwicklung der Algorithmen 5
2.5.1 Schneeflocke 5
2.5.2 Pfeilspitze 6
2.5.3 Drachenkurve 6
2.6 Testen 6
3. Resultate 6
3.1 Grafisch dargestellte Fraktale 6
3.2 Effizienz 7
3.3 Features des Main-Programms? Oder zu einleitung?? 7
4. Diskussion 7
Flächedrachenkurve 7
4.1 Optimierungsmöglichkeiten 7
4.2 Rückblick auf das Problem 8
5. Nachwort? 8
6 Quellenverzeichnis 8
7. Abbildungsverzeichnis 8
8. Tabellenverzeichnis 8
9. Quellcodeverzeichnis 8
10. Anhang 8
10.1 Weitere Diagramme 8
10.2 Gesamter Quellcode 8

1. Einleitung

1.1 Was sind Fraktale?

Fraktale sind Geometrische Gebilde, die sich in nicht ganzzahligen („fraktalen“) Dimensionen befinden. B.B. 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), kein Datum).

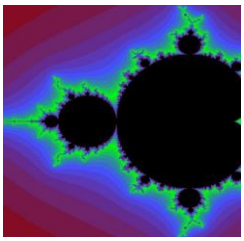


Abb. 1 Mandelbrot Fraktal (Vater der Fraktale: Mandelbrot bewies die Schönheit der Mathematik - WELT, 2010)

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 Problem, das in diesem Projekt untersucht wurde, soll bestimmte Fraktale, wie die „Schneeflocke“, die „Pfeilspitze“ und die „Drachenkurve“ darstellen, bzw. zeichnen. Das Vorgehen beim Lösen des Problems wird ebenfalls in dieser Portfolioarbeit dargestellt.

Kommentiert [JS1]: Quellen?

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*.

Kommentiert [JS2]: Erklären

Kommentiert [JS3]: Quelle?

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.

Kommentiert [JS4]: Mehr schreiben

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 Funktionalitäten 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:

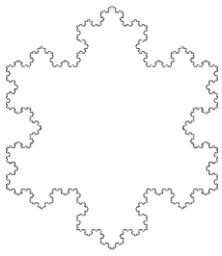


Abb. 3 Schneeflocke bei Stufe 7

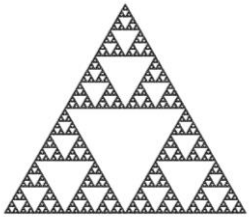


Abb. 4 Pfeilspitze bei Stufe 9

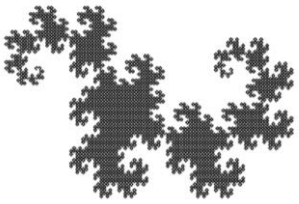


Abb. 2 Drachenkurve bei Stufe 13

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 einem Total von ca. 500 Linien Quellcode. Eine gesamte Übersicht 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 wachsender Grösse 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 Kapitel 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 Forderungen.

2.1 Programmiersprache

Wie 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).

Kommentiert [JS5]: Weglassen?

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, 2021). Die Bibliothek bietet eine Zeichenfläche und Methoden an, darauf zu zeichnen. 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.

Kommentiert [JS6]: Quelle?

2.2.2 Flanagan

Für das Einlesen der gewünschten Kurve, Stufe und Dateiname 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), 2010) implementiert. Mit ihr 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.

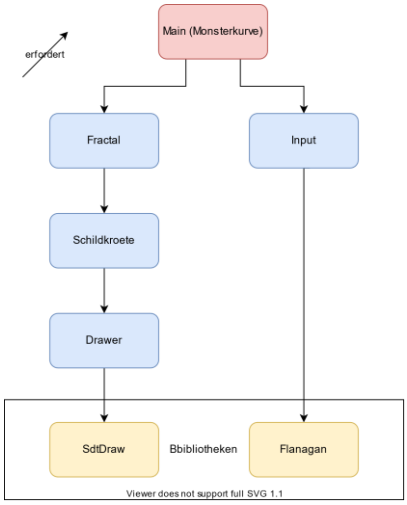


Abb. 5 Modularisierung des Programms

Dieses Programm habe ich folgendermassen modularisiert:

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*.

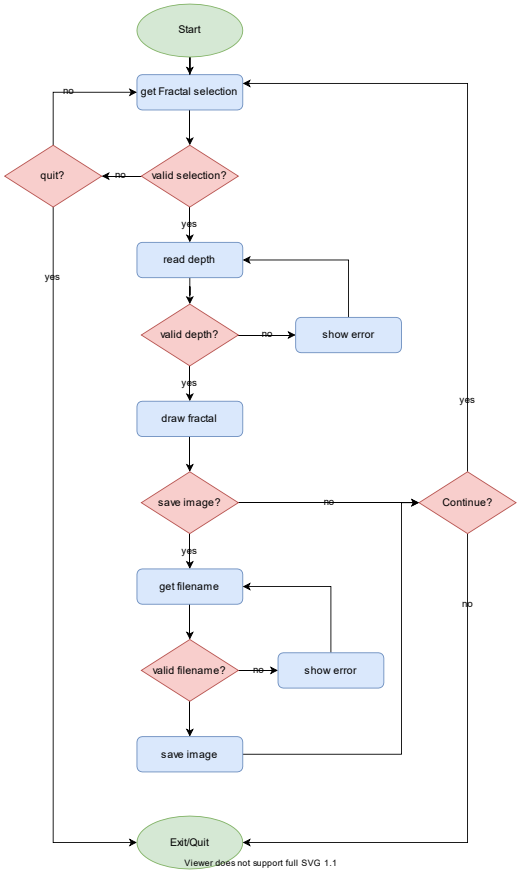


Abb. 6 Grobalgorithmus des Main Programms

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

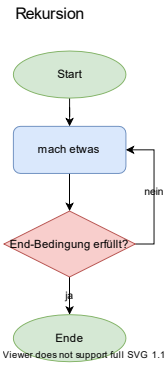


Abb. 7 Ablaufsprinzip der Rekursion

Bei allen drei Fraktalen beruht der Grobalgorithmus auf das allgemeine 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).

2.4 Entwicklung der Module

Ablaufdiagramme, Beschreibung variablen

2.5 Entwicklung der Algorithmen

Wie erwähnt, verwenden alle drei Algorithmen der Fraktale die Rekursion. Bei den drei Algorithmen ist der Parameter *depth (int)* und *length (double)* gegeben. Sie manipulieren die Richtung der *Schildkroete* und rufen ihre eigene Methode wieder auf mit einer niedrigeren Stufe und einer verkleinerten Strecke. Wenn die Stufe 0 ist, 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 mehr und kleinere gerade Strecken aufweist. Diese kleineren 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

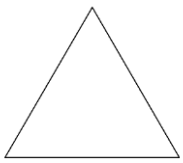


Abb. 10 Schneeflocke 0

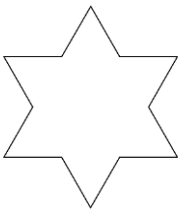


Abb. 9 Schneeflocke 1

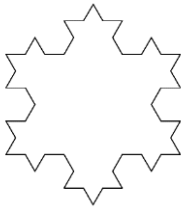


Abb. 8 Schneeflocke 2

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 Dreicks ersetzt wird (siehe Abb. 10, Abb. 9, Abb. 8).

INSERT_CODE

2.5.2 Pfeilspitze

Abb. 13 Pfeilspitze 0

Abb. 12 Pfeilspitze 1

Abb. 11 Pfeilspitze 2

Die Pfeilspitzenkurve (auch Sierpinski-Dreieck genannt) hat eine gerade Strecke als *Initiator*. Ihr *Generator* ist eine Dach-förmige Figur mit 60 Grade 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 ein 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 kleine als auch katastrophale Fehler zu vermeiden. Dabei ist es wichtig, nicht nur die optimalen Eingabe-Werte des Benutzers zu erwarten, sondern auch die komischsten Werte. Diese App wurde mit allmöglichen Werten getestet und so gebaut, dass sie im Falle falscher Eingaben, nützliche an den Benutzer Fehlermeldungen gibt.

Ebenfalls wichtig ist die Effizienz der Algorithmen. Diese wurde gemessen und die Resultate werden in 3.2 Effizienz besprochen und analysiert. Dafür wurden die Zeit und wie oft die *Schildkroete* sich bewegen muss Betracht gezogen.

3. Resultate

3.1 Grafisch dargestellte Fraktale

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

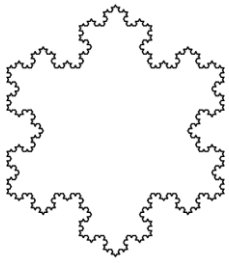


Abb. 16 Schneeflocke 4

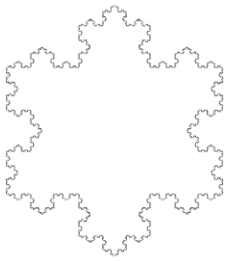


Abb. 14 Schneeflocke 6

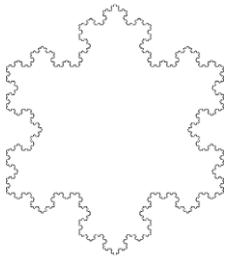


Abb. 15 Schneeflocke 8

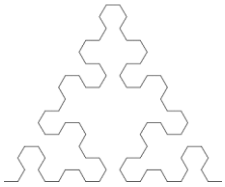


Abb. 21 Pfeilspitze 4

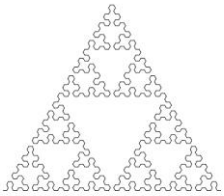


Abb. 17 Pfeilspitze 6

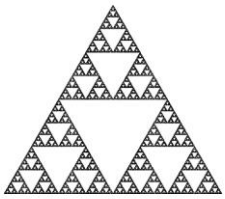


Abb. 22 Pfeilspitze 9

Abb. 20 Drachenkurve 6

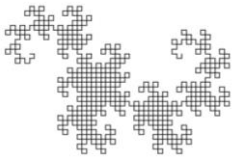


Abb. 19 Drachenkurve 10

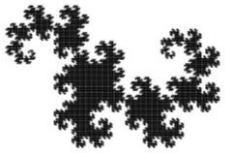


Abb. 18 Drachenkurve 14

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 Sekunde oder gar mehrere Minuten dauert, muss darauf geachtet werden. Bei diesem 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 Darstellung 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 Bewegungen der Schildkroete ist und *n* die Stufe, so ist bei der Schneeflocke $s = 4^n$, bei der Pfeilspitze $s = 3^n$ und bei der Drachenkurve $s = 2^n$. Die Laufzeitdauer 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 $s = 4^n$. Bei der **Pfeilspitze** besteht der Generator aus drei Strecken, also müssen pro Stufe dreimal so viele Strecken gezeichnet werden, deshalb die Formel $s = 3^n$. 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 Laufzeitdauer zu erhalten. Damit muss man sich einfach abfinden.

Kommentiert [JS7]: Weglassen? Anders schreiben?

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
Abb. 2 Drachenkurve bei Stufe 13 2
Abb. 3 Schneeflocke bei Stufe 7 2
Abb. 4 Pfeilspitze bei Stufe 9 2
Abb. 5 Modularisierung des Programms 4
Abb. 6 Grobalgorithmus des Main Programms 4
Abb. 7 Ablaufsprinzip der Rekursion 5
Abb. 10 Schneeflocke 2 5
Abb. 8 Schneeflocke 1 5
Abb. 9 Schneeflocke 0 5

8. Tabellenverzeichnis

9. Quellcodeverzeichnis

10. Anhang

10.1 Weitere Diagramme

10.2 Gesamter Quellcode