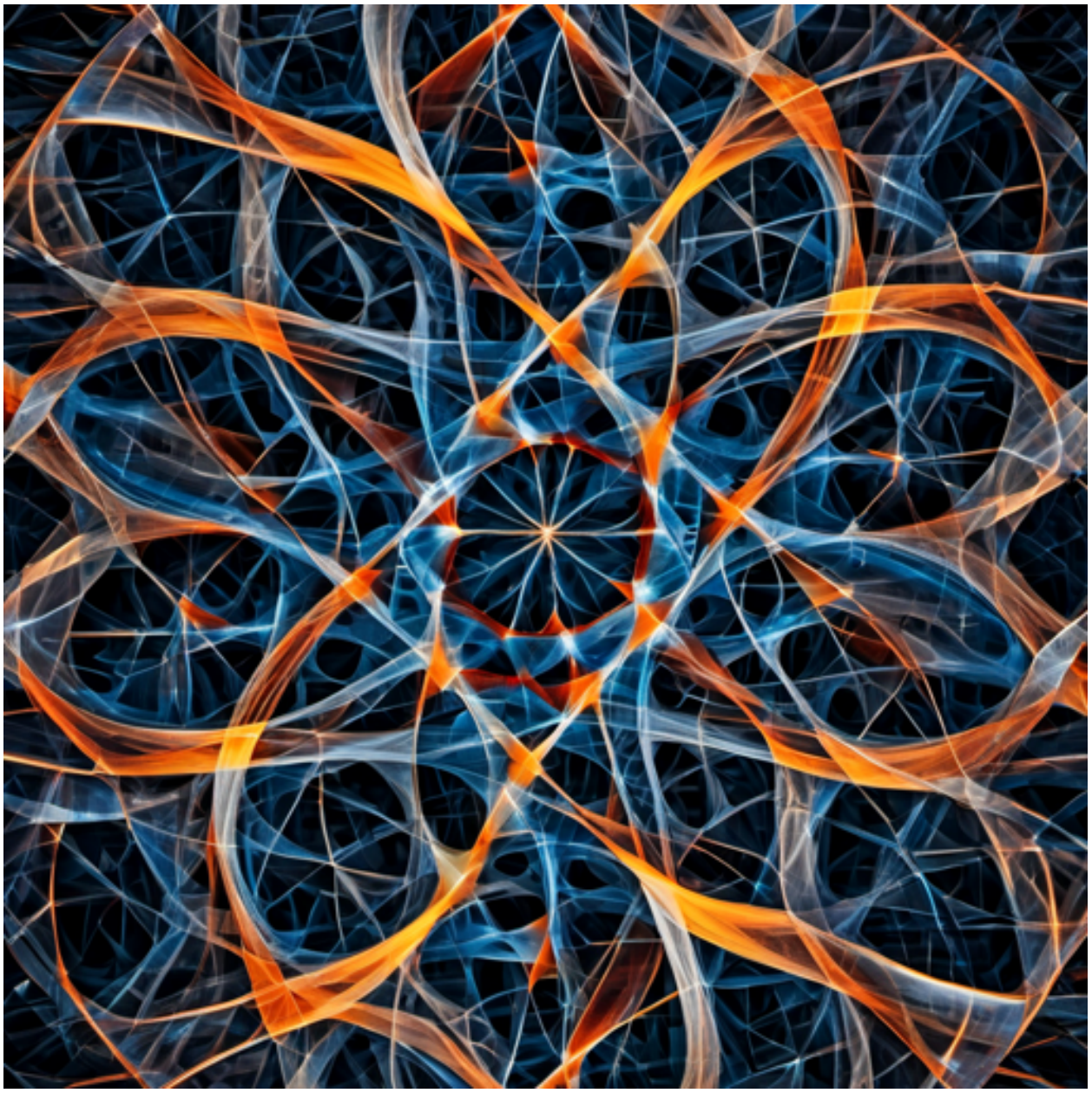


PolyPlotter

Sarina Lusti

Amanda Walser

Inhaltsverzeichnis

PolyPlotter

Planung

Zu Beginn des Moduls haben wir mit der Projektarbeit begonnen, da die anderen Aufgaben für das Modul noch nicht vollständig bereit waren. Dabei haben wir uns darauf geeinigt, ein Tamagotchi mit Front- und Backend zu programmieren. Dieses Projekt hatten wir so gut es zu dieser Zeit möglich war geplant. Unter anderem haben wir beschlossen, die Aufgabenteilung für das Projekt in Front- und Backend aufzuteilen. Anschliessend haben wir uns darauf fokussiert, die Aufgaben zu lösen. Gegen Ende des Moduls haben wir gemerkt, dass die Aufgaben einen grösseren Umfang hatten, als wir zu Beginn des Modules dachten. Dies führte dazu, dass wir uns entschieden haben, das Tamagotchi Projekt nicht umzusetzen, da dies aus zeitlichen Gründen nicht möglich gewesen wäre.

Das PolyPlotter Projekt ist ein Projekt, welches Frau Lusti bereits in einem vorherigen Modul umgesetzt hat. Dadurch mussten wir uns nicht darum kümmern, ein komplett neues Projekt aufzusetzen und zu implementieren. Zudem konnten wir uns auf das Testing, das Test Driven Development und die Code Reviews fokussieren.

Das Projekt bestand vor unserer Erweiterung aus Funktionen, mit denen man aus bestimmten Zahlenmengen, wie der Mandelbrot-Menge, einen Graph plotten konnte. Das Projekt hatte noch keine Tests, diese haben wir alle in diesem Modul hinzugefügt. Wir haben uns entschieden, ein neues Feature zu implementieren, damit wir das Test Driven Development anhand des Projektes umsetzen konnten. Dabei haben wir uns entschieden, eine neue Funktion hinzuzufügen, mit der man individuelle Graphen berechnen und erstellen kann. Dazu gibt der User eine mathematische Funktion ein und der PolyPlotter erstellt den dazugehörigen Graphen.

Wir haben zusammen die Anforderungen an das neue Feature besprochen und daraus konkrete Aufgaben definiert. Anschliessend haben wir diese Aufgaben untereinander aufgeteilt. Da die Zeit für die Umsetzung des Features und des Testings knapp war, haben wir uns auch damit auseinandergesetzt, wie wir miteinander kommunizieren und die Arbeitsschritte, welche teilweise Abhängigkeiten hatten, koordinieren. Das Ergebnis davon war, dass Frau Lusti den Interpreter und Frau Walser die grafischen Elemente und die Berechnung des Graphen implementieren. Zudem waren beide dafür verantwortlich, ihren Code zu testen und ihre Erfahrungen in der Dokumentation festzuhalten. Anschliessend haben wir in GitHub zudem eine CI/CD Pipeline erstellt, um unsere Tests zu automatisieren.

Visualisierung der Architektur

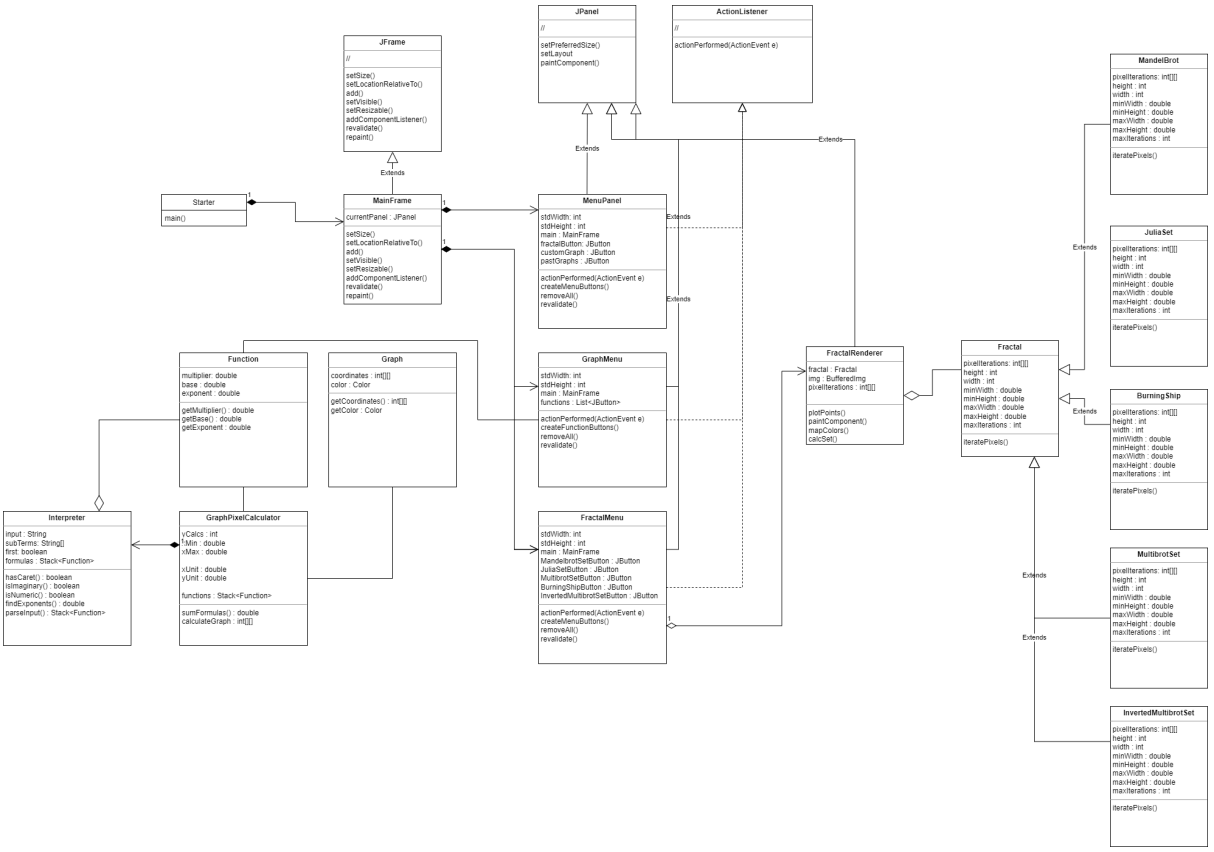
Unser Projekt ist so aufgebaut, dass es ein Hauptmenü gibt. Dort kann man das Projekt schliessen, das Fraktal-Menü oder das Graph-Menü öffnen. Das Fraktal-Menü wurde von Frau Lusti bereits vor diesem Modul implementiert. Es besteht aus mathematischen Zahlen Sätzen wie dem Mandelbrot-Satz.

In diesem Modul implementiert haben wir das Graph Menü. Darin kann man eine mathematische Funktion eingeben und anhand dessen wird dann ein passender Graph

erstellt. Dafür brauchen wir diverse Klassen: um sicherzustellen, dass der Input valide ist; zum Berechnen des Graphen; zum Erstellen der Grafik.

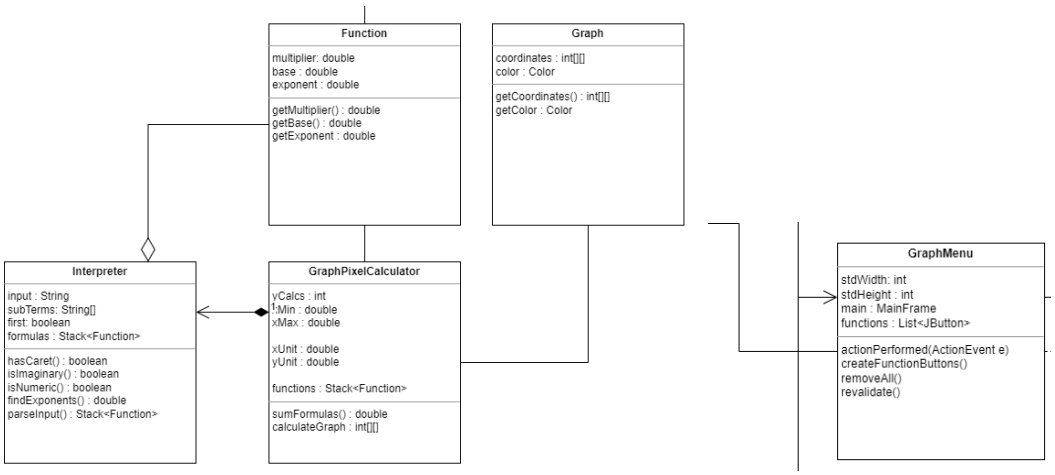
UML gesamtes Projekt

Unten sieht man das UML des kompletten Projektes.



UML Fokus Custom Graph

Hier sieht man den Ausschnitt, den wir für dieses Modul implementiert haben.



Testkonzept

Einführung

Unsere Applikation ist ein PolyPlotter, der anhand der Auswahl eines spezifischen Fraktals oder der Eingabe einer mathematischen Funktion einen Graphen erstellt. Ein Teil des Projekts wurde von Frau Lusti bereits in einem anderen Modul implementiert. Doch wir haben das komplette Testing und ein neues Feature hinzugefügt. Das Feature, welches wir hinzugefügt haben, ist das Erstellen eines Graphens anhand einer mathematischen Funktion, die der User eingeben kann.

Test Objekte und Features

MenuPanel

- Exit & Save Button
    - Bei Knopfdruck wird das Programm beendet
- Fractal Button
    - Der Fractal Knopf existiert im UI
- Navigates to Fractal Menu
    - Bei Knopfdruck wird der User zum FractalMenu weitergeleitet

FractalMenu

- Navigate Back to Main Menu
    - Bei Knopfdruck wird man zum Main Menu zurück geleitet
- Mandel Brot Set Button
    - Bei Knopfdruck wird das Mandel Brot Set ausgewählt und der passende Graph dazu wird erstellt
- Julia Set Button
    - Bei Knopfdruck wird das Julia Set ausgewählt und der passende Graph dazu wird erstellt
- Burning Ship Button
    - Bei Knopfdruck wird das Burning Ship Set ausgewählt und der passende Graph dazu wird erstellt
- Tricorn Set Button
    - Bei Knopfdruck wird das Tricorn Set ausgewählt und der passende Graph dazu wird erstellt
- Multi Brot Set Button
    - Bei Knopfdruck wird das Multi Brot Set ausgewählt und der passende Graph dazu wird erstellt
- Inverted Multi Brot Set Button
    - Bei Knopfdruck wird das Inverted Multi Brot Set ausgewählt und der passende Graph dazu wird erstellt

FractalRenderer

- Go Back Button
    - Der Fractal Renderer erstellt einen Go Back Button

- FractalRenderer
    - Der FractalRenderer wird erstellt und grafisch dargestellt
- Navigate Back to Main Menu
    - Wird der Go Back Button gedrückt, wird der User zum Main Menu geleitet

Validator

- Valider Input mit einer Variablen
    - Der Input ist eine valide mathematische Funktion mit einer Variablen
- Invalider Input mit mehreren Variablen
    - Der Input enthält mehrere Variablen und ist somit ungültig
- Get Input
    - Den korrekten Input aus dem Validator holen
- Set Input
    - Den korrekten Input im Validator setzen
- Remove Functions
    - Mathematische Funktionen wie sin() aus dem Input-String entfernen

Interpreter

- ConvertToReversePolishNotation mit validem Input
    - Der eingegebene String wird aufgeteilt und nach Zeichen sortiert
- ConvertToReversePolishNotation mit invalidem Input -> Klammern
    - Die ParenthesesMismatchException wird aufgerufen und der Code läuft nicht weiter

DoubleStack

- addsOnPush
    - Ein Wert wird hinzugefügt
- pop
    - Ein Element wird gelöscht
- isEmpty
    - Der Stack ist leer
- isFull
    - Der Stack ist voll

StringDeterminer

- isFunction
    - Ein Zeichen eines Strings ist eine Funktion
- isSeparator
    - Ein Zeichen eines Strings ist ein Separator
- isNumerical_false
    - Ein Zeichen eines Strings ist keine Zahl, obwohl die isNumerical Methode aufgeufen wird, deswegen ist dieser Testfall inkorrekt
- isOperator_false
    - Ein Zeichen eines Strings ist ein invalider Operator, deswegen ist dieser Testfall inkorrekt

StringStack

- addsOnPush
    - Ein Wert wird hinzugefügt
- pop
    - Ein Element wird gelöscht
- isEmpty
    - Der Stack ist leer
- isFull
    - Der Stack ist voll

Tokenizer

- getTokens
    - Teilt einen String in einzelne Zeichen auf
- getVariable
    - Holt die Variable aus einem String

GraphPixelCalculator

- Ein Pixel Array erstellen
    - In dem Pixel Array sollen die Pixel so abgespeichert sein, dass man sie anschliessend verwenden kann, um die Grafik zu printen. Der Graph wird aus den eingegebenen Werten erstellt.

Fehler Kriterien

- Leichte Fehler:
    - Wenn eine Funktionalität nicht optimal läuft, aber der User nichts davon mitkriegt
- Mittelschwere Fehler:
    - Wenn eine Funktionalität so stark beeinflusst ist, dass sie noch funktioniert, aber den User trotzdem negativ beeinflusst z.B. eine deutlich höhere Zeit beim Berechnen des Graphen
- Schwere Fehler:
    - Wenn eine Funktionalität im Code komplett nicht mehr korrekt funktioniert

Test Deliverables

- Tools
    - Unit Tests in Gradle und resultierender Coverage Bericht
    - End-to-End Tests in Gradle
    - CI/CD Pipeline auf GitHub

Testing Aufgaben

Wir haben uns in unserem Projekt hauptsächlich auf das Unit Testing fokussiert, da unsere Applikation dadurch am einfachsten getestet werden kann und wir die höchste Code Abdeckung erreichen können. Zusätzlich haben wir auch noch einen End-to-End-Test implementiert, um sicherzustellen, dass in unserem Programm die geforderten Abläufe und das Zusammenspiel der Komponenten funktioniert. Diese End-to-End-Tests sind etwas

anders, als man sie von einer klassischen Webapplikation kennt, da wir unsere grafischen Elemente mit Java Swing erstellt haben.
Grundsätzlich wurde trotz dessen eine User Journey durchgespielt, die sich vor allem mit dem Navigieren und der Eingabe einer Funktion beschäftigt. Somit sind verschiedene Schnittstellen abgedeckt.

Testumgebungs Anforderungen

Um unser Projekt und die Tests auszuführen, muss man das Repository clonen. Ausserdem muss man Java und Gradle installiert haben. Die Unit Tests in unserem Projekt muss man dann lokal ausführen.
Die von uns verwendete Java Version ist 17.

Zeitplan

Die Unit Tests, welche wir für den bereits vorhandenen Code geschrieben haben, haben wir geschrieben, als der Code schon existiert. Bei den grafischen Komponenten haben wir die Tests zu einem grossen Teil auch vor dem Code geschrieben. Beim restlichen Code des neuen Features haben wir das Test Driven Development angewendet. Das heisst, dass wir zuerst die Tests geschrieben haben und anschliessend den Code. Dadurch konnten wir immer überprüfen, ob unser Code funktioniert und macht, was er soll oder eben nicht.

Reflektion zu TDD und Code Reviews

Test Driven Development

Das Test Driven Development hat Vorteile, wie dass man sich bei der Implementation des Codes nicht so sehr darauf fokussieren muss, ob der Code die korrekte Funktionalität hat. Man kann also während des Programmierens die Tests laufen lassen und sieht, ob die gewünschte Funktionalität vorhanden ist oder nicht. Doch das Test Driven Development ist unserer Erfahrung nach aufwändiger, da man sich zwei Mal überlegen muss, welche Funktionalität notwendig ist. Für uns ist es einfacher, anhand von existierendem Code einen Test zu schreiben, als ein Code, der zu einem bereits bestehenden Test passt. Wir vermuten, dass das so ist, weil wir es uns gewohnt sind, Code zu schreiben und diesen zu testen, aber das TDD haben wir in unserem Alltag noch nicht so ausgiebig verwendet. Dennoch gilt als großes Pro zu erwähnen, dass dieses Üben von TDD Denkanstöße gibt, die es einfacher machen, verschiedene Perspektiven des Code Developments zu sehen, von denen man natürlich auch profitieren kann. Das gilt vor allem, wenn es um Klassendesign geht. Wenn man die Seite des Testings besser kennt, profitiert auch das Implementationsdesign davon und steigert somit die Code Qualität.

Code Reviews

Code Reviews kennen wir aus unserem Alltag bereits sehr gut. Wir finden es sehr sinnvoll, da alle Personen, die an einem Projekt sind dadurch automatisch zu verstehen geben, dass sie mit dem Code zufrieden sind und ansonsten einen Kommentar hinterlassen. Dieser Kommentar ist dann bereits schon an der richtigen Stelle im Code und man muss nicht lange suchen. Wenn man mit dem Kommentar der anderen Person nicht einverstanden ist,

kann man seine Meinung auch direkt unter dem Kommentar wiedergeben, so dass die Konversationen zum Code alle am gleichen Ort sind und man alles gut nachvollziehen kann.