HAW HAMBURG

BACHELOR THESIS
Flemming Grabowski

Sketch-basierte Generierung von fiktionalen Landkarten

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Flemming Grabowski

Sketch-basierte Generierung von fiktionalen Landkarten

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Philipp Jenke
Zweitgutachter: Prof. Dr. Peer Stelldinger

Eingereicht am: 06. März 2025

Flemming Grabowski

Thema der Arbeit

Sketch-basierte Generierung von fiktionalen Landkarten

Stichworte

Prozedurale Content Generierung, Fiktionale Landkarten, Sketch-basierte Generierung, Geometrische Algorithmen, Regelbasierte Verfahren, Noise-basierte Verfahren

Kurzzusammenfassung

Diese Arbeit stellt die Entwicklung eines prototypischen Systems zur sketch-basierten Generierung fiktionaler Landkarten vor, bei dem geometrische Algorithmen wie die Delaunay-Triangulation mit prozeduralen Methoden kombiniert werden. Basierend auf Skizzen werden Landmassen segmentiert, Regionen durch Icons definiert und mittels Noise-basierter sowie regelbasierter Verfahren detaillierte Landschaften erzeugt. Die Evaluation zeigt, dass durch die gezielte Kombination dieser Methoden eine kohärente und intuitiv steuerbare Kartengenerierung möglich ist. Gleichzeitig wurden Herausforderungen identifiziert, insbesondere in der Handhabung ungenauer Eingaben und der Echtzeitperformance. Das System ermöglicht modular erweiterbare Generierungsstrategien und sorgt für natürliche Übergänge zwischen Biomen.

Flemming Grabowski

Title of Thesis

Sketch-based Generation of Fictional Maps

Keywords

Procedural Content Generation, Fictional Maps, Sketch-based Generation, Geometric Algorithms, Rule-based Methods, Noise-based Methods

Abstract

This thesis presents the development of a prototype system for sketch-based generation of fictional maps, combining geometric algorithms such as Delaunay triangulation with

procedural methods. Based on sketches, landmasses are segmented, regions are defined through icons, and detailed landscapes are generated using noise-based and rule-based techniques. The evaluation demonstrates that a targeted combination of these methods enables coherent and intuitively controllable map generation. At the same time, challenges were identified, particularly in handling imprecise inputs and ensuring real-time performance. The system allows for modularly extensible generation strategies and ensures natural transitions between biomes.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Die Erstellung fiktionaler Landkarten spielt eine zentrale Rolle in vielen kreativen Bereichen, darunter Videospiele, Pen-and-Paper-Rollenspiele und Fantasy-Literatur. Traditionell erfordert die Gestaltung solcher Karten viel künstlerisches Geschick und Zeitaufwand. Insbesondere für Personen ohne zeichnerische Vorerfahrung kann dies eine erhebliche Hürde darstellen.

Eine Möglichkeit, diesen Prozess zu erleichtern, liegt in der Verwendung von prozeduralen Algorithmen, die auf Grundlage bestimmter Eingaben eine detailreiche und kohärente Karte generieren. Bisher existierende Systeme zur automatisierten Kartenerstellung bieten jedoch oft wenig interaktive Gestaltungsmöglichkeiten oder sind stark an bestimmte Darstellungsstile gebunden.

Diese Arbeit untersucht den Ansatz der sketch-basierten Generierung fiktionaler Landkarten. Dabei können Nutzer durch einfache Skizzen und Markierungen den grundlegenden Aufbau ihrer Karte definieren, während Algorithmen die Skizze analysieren und weiterverarbeiten, um eine detailreiche, kohärente und optisch ansprechende Karte zu erzeugen.

Die zentrale Forschungsfrage lautet dabei: *Lässt sich eine strategische Kombination algorithmisch unterschiedlicher Verfahren so gestalten, dass trotz ihrer strukturellen Divergenz eine visuell kohärente und intuitiv steuerbare sketch-basierte Generierung fiktionaler Landkarten ermöglicht wird?*

1.2 Ziele

Ziel dieser Arbeit ist die Entwicklung eines prototypischen Systems zur sketch-basierten Generierung von fiktionalen Landkarten. Das System soll es ermöglichen, durch einfache Strichzeichnungen und Markierungen geografische Strukturen wie Küstenlinien, Gebirge oder Wälder intuitiv zu definieren. Diese Eingaben werden anschließend durch eine Kombination aus geometrischen Algorithmen, regelbasierten Verfahren und noise-basierten Methoden weiterverarbeitet.

Die zentralen Teilziele sind:

- **Benutzerfreundlichkeit**: Das System soll eine intuitive Zeicheneingabe ermöglichen und auch ungenaue Skizzen zuverlässig interpretieren.
- **Modulare Generierungsstrategien**: Unterschiedliche algorithmische Verfahren sollen trotz ihrer strukturellen Divergenz zu einem kohärenten System integriert und flexibel kombiniert werden können, um eine hohe Anpassungsfähigkeit zu gewährleisten.
- **Natürlich wirkende Ergebnisse**: Die generierten Karten sollen visuell ansprechend sein und organische Übergänge zwischen Landschaftstypen enthalten.
- **Effizienz**: Die Generierung soll in einer angemessenen Zeit erfolgen, um eine interaktive Nutzung zu ermöglichen.

1.3 Aufbau der Arbeit

Die vorliegende Arbeit ist in mehrere Kapitel unterteilt, die schrittweise von den theoretischen Grundlagen bis zur praktischen Umsetzung und Evaluation des entwickelten Systems führen.

Kapitel 2 beschreibt die theoretischen Grundlagen der verwendeten Algorithmen und Methoden. Dazu gehören geometrische Verfahren wie die Delaunay-Triangulation und Voronoi-Diagramme sowie verschiedene Ansätze der prozeduralen Content Generierung.

Kapitel 3 gibt einen Überblick über bestehende Systeme zur Generierung von Landkarten mit einem Fokus auf sketch-basierte und prozedurale Ansätze. Dazu werden verschiedene Methoden analysiert und miteinander verglichen, um deren Stärken und Schwächen herauszuarbeiten.

Kapitel 4 entwickelt darauf aufbauend ein Konzept für das eigene System. Hier werden die funktionalen und nicht-funktionalen Anforderungen definiert, der grundlegende Verarbeitungsprozess skizziert und das methodische Vorgehen beschrieben.

Kapitel 5 behandelt die technische Umsetzung, einschließlich der verwendeten Technologien, der Systemarchitektur und der Implementierung der einzelnen Komponenten.

Kapitel 6 bewertet die verwendete Methodik, evaluiert die Qualität der generierten Karten anhand von visuellen und technischen Kriterien, reflektiert die Herausforderungen des Systems und gibt einen Ausblick auf mögliche Erweiterungen.

Abschließend fasst Kapitel 7 die Ergebnisse zusammen.

2 Grundlagen

2.1 Geometrische Algorithmen

Im Kontext der sketch-basierten Generierung spielen geometrische Algorithmen eine wichtige Rolle. Sie ermöglichen es, die Benutzereingabe wie vom Nutzer beabsichtigt einzulesen und in geeigneten Datenstrukturen zu halten. Somit können einfache Benutzereingaben wie das Gedrückthalten einer Maustaste und das gleichzeitige Bewegen der Maus zu einer komplexen Struktur wie einem Dreiecksnetz umgewandelt werden, welches dann wiederum genutzt werden kann, um effizient mit der Eingabe zu arbeiten.

In diesem Abschnitt werden drei wesentliche geometrische Algorithmen vorgestellt, die in der Computergrafik und der algorithmischen Geometrie weit verbreitet sind:

- Der Bresenham-Algorithmus, der eine effiziente Methode zur rasterbasierten Linienzeichnung bereitstellt und dabei nur Ganzzahloperationen verwendet.
- Die Delaunay-Triangulation, die eine optimale Zerlegung einer Menge von Punkten in Dreiecke ermöglicht, dabei aber spitze Winkel vermeidet.
- Das Voronoi-Diagramm, welches eine Ebene in mehrere Regionen unterteilt.

2.1.1 Bresenham

Wenn zwei Punkte, die irgendwo auf einem Computerbildschirm gesetzt werden, verbunden werden sollen, indem eine Linie gezeichnet wird, dann muss diese durch die Auswahl der nächstgelegenen Pixel möglichst präzise und effizient dargestellt werden. Der Bresenham-Algorithmus löst dieses Problem, indem er entscheidet, welcher Pixel dem idealen Linienverlauf am nächsten liegt, und dabei nur Ganzzahlen sowie einfache Additionen, Subtraktionen und Vergleiche verwendet [2]. Der Algorithmus funktioniert wie folgt:

Als Erstes werden die Differenzen in x- und y-Richtung zwischen Startpunkt und Endpunkt berechnet (dx und dy). Anschließend wird die „schnelle Richtung“ bestimmt – also die Achse, entlang derer die Linie am stärksten ansteigt. Der Algorithmus schreitet dann Pixel für Pixel in dieser schnellen Richtung voran und entscheidet bei jedem Schritt anhand einer Fehlervariable D, ob ein zusätzlicher Schritt in der langsamen Richtung nötig ist, um die ideale Linie möglichst exakt nachzuziehen. Der initiale Fehler wird als

$$D = 2 \cdot dy - dx$$

berechnet. In der Hauptschleife wird dann für jeden Pixel folgendes durchgeführt:

- Der aktuelle Pixel wird gezeichnet.
- Es wird überprüft, ob der Zielpunkt erreicht wurde.
- D wird um $2 \cdot dy$ inkrementiert.
- Falls $D \geq 0$, wird zusätzlich ein Schritt in der langsamen Richtung ausgeführt und D um $2 \cdot dx$ dekrementiert.

Der Algorithmus ist sehr effizient, da:

- Er nur ganzzahlige Operationen verwendet.
- Nur einfache Additionen, Subtraktionen, Vergleiche und konstante Multiplikationen benötigt.
- Die Laufzeit $O(\max(dx, dy))$ beträgt, die Schleife also nur so oft durchlaufen wird, wie die längste Achsendifferenz vorgibt.

Der Algorithmus lässt sich aufgrund der genannten Aspekte hervorragend in jegliche Systeme implementieren, in denen per Hand Linien gezeichnet werden und in denen für ein möglichst natürliches Ergebnis interpoliert werden muss.

2.1.2 Delaunay-Triangulation

Die Delaunay-Triangulation ist ein fundamentales Konzept im Bereich der algorithmischen Geometrie. Sie wurde ursprünglich von Boris Delone eingeführt [3] und ermöglicht eine effiziente Zerlegung einer Punktmenge P in eine Dreiecksmenge T. Die Menge T zeichnet sich durch die Eigenschaft aus, dass der Umkreis jedes Dreiecks keine weiteren Punkte aus P enthält. Somit entsteht eine Triangulation, die die kleinsten Innenwinkel über alle Dreiecke aus T maximiert, es werden also sehr spitze Winkel vermieden [12].

Es gibt unterschiedliche Ansätze für das Erstellen einer solchen Triangulation. Ein iterativer Ansatz wäre der Bowyer-Watson-Algorithmus, welcher über die Punktmenge P iteriert und dabei die Punkte sukzessive zur Triangulation hinzufügt, aber dabei die genannte Eigenschaft beibehält [1].

Die Delaunay-Triangulation lässt sich in den folgenden Schritten zusammenfassen:

- **Initialisierung:** Ein Dreieck wird erstellt, das alle Punkte aus P umfasst.
- **Überprüfung der Delaunay-Eigenschaft:**
  - Für jeden Punkt p_i aus P wird überprüft, welche Dreiecke aus der Triangulation die Delaunay-Eigenschaft verletzen.
  - Falls ein Dreieck die Delaunay-Eigenschaft verletzt, wird es markiert.
- **Entfernung der Kanten:** Die Kanten der markierten Dreiecke werden entfernt, was zu einem "Loch"führt.
- **Verschließen des Lochs:** Das Loch wird durch das Verbinden von p_i mit den Eckpunkten der entfernten Dreiecke geschlossen.
- **Entfernung des Initialisierungs-Dreiecks:** Das Initialisierungs-Dreieck wird entfernt, und die Delaunay-Triangulation ist vollständig.

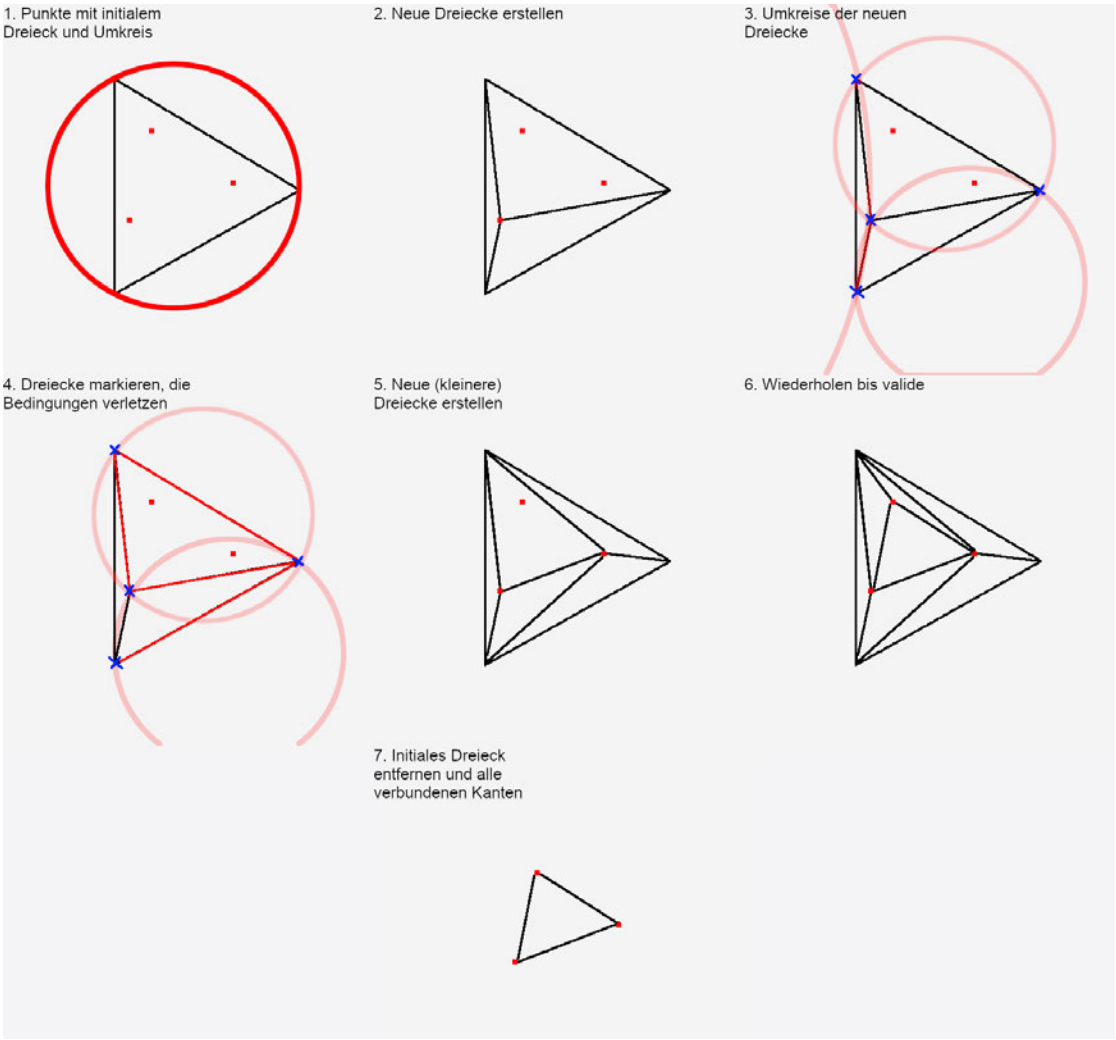

Abbildung 2.1: Visualisierung der Delaunay-Triangulation einer Punktmenge.
[1]

Heutzutage findet man Delaunay-Triangulationen in einer Vielzahl von Anwendungsgebieten. Darunter fallen beispielsweise die Analyse medizinischer Bilder zur Hautkrebserkennung sowie die Erzeugung realistischer 3D-Modelle [6].

[1] Eigene Abbildung, nachempfunden basierend auf der Quelle: https://www.gorillasun.de/blog/bowyer-watson-algorithm-for-delaunay-triangulation/.

2.1.3 Voronoi-Diagramme

Ein Voronoi-Diagramm, ursprünglich eingeführt von Georgy Voronoi [22], ist ein grundlegendes geometrisches Konstrukt, welches eine Ebene in Regionen unterteilt. Diese Unterteilung entsteht durch die geografische Lage von Punkten aus einer Punktmenge P. Jede Region, auch Voronoi-Zelle genannt, enthält genau einen Punkt p aus P und umfasst alle Punkte der Ebene, die näher an p liegen als an allen anderen Punkten aus P. Somit besitzt jeder Punkt, der auf einer Kante zweier Voronoi-Zellen liegt, den gleichen Abstand zu den beiden erzeugenden Punkten.

Das Voronoi-Diagramm ist dual zur bereits vorgestellten Delaunay-Triangulierung, was bedeutet, dass sich aus der einen Struktur immer die jeweils andere ableiten lässt. Die Dualität zeigt sich darin, dass zwei Punkte genau dann durch eine Delaunay-Kante verbunden sind, wenn ihre zugehörigen Voronoi-Regionen eine gemeinsame Kante teilen [13].

Voronoi-Diagramme können genutzt werden, um möglichst natürliche Grenzen innerhalb eines Bereiches, wie dem Umriss einer Landmasse, zu generieren, um diese in distinkte Regionen zu unterteilen. Die Voronoi-Zellen erzeugen organisch wirkende Grenzverläufe, die den natürlichen Gegebenheiten einer Landkarte ähneln.

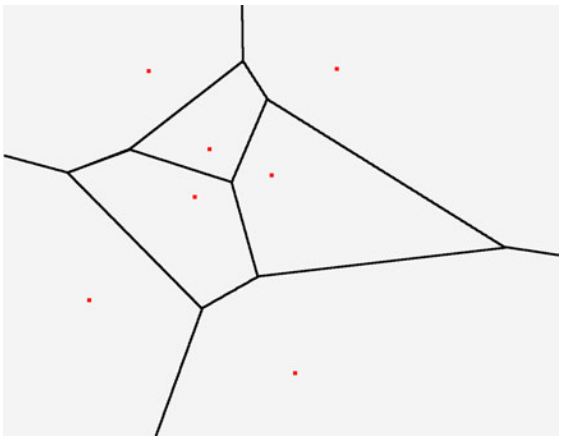

Abbildung 2.2: Visualisierung des Voronoi-Diagramms für eine Punktmenge.
[2]

[2] Eigene Abbildung, nachempfunden basierend auf der Quelle: https://de.wikipedia.org/wiki/Voronoi-Diagramm.

2.2 Clustering und Formerkennung

Die Grundlagen des Clusterings und der Formerkennung, auf die im Folgenden eingegangen wird, können nützlich sein, um zugrundeliegende Datenstrukturen zu analysieren, zu gruppieren und anzupassen. So macht es Clustering im Kontext von Skizzen beispielsweise möglich, lokale Eigenschaften der jeweiligen Skizze getrennt zu behandeln. Die Formerkennung bietet die Möglichkeit, eine Skizze parametrisierbar zu analysieren und zu verfeinern.

2.2.1 Density-Based Spatial Clustering of Applications with Noise

Der Density-Based Spatial Clustering of Applications with Noise (DBSCAN) Algorithmus, entwickelt von Ester et al., ist ein weit verbreiteter Algorithmus für Data-Mining-Zwecke [7]. Im Bereich des Data Minings müssen Datenpunkte oft kategorisiert beziehungsweise gruppiert werden. Die Anforderungen an einen solchen Algorithmus sind meistens: Effizienz, die Bedingung, dass dieser trotz wenig Vorwissen über die Domäne funktioniert, und vor allem die Fähigkeit, Cluster unterschiedlichster Formen zu erkennen. Der DBSCAN-Algorithmus erfüllt diese Bedingungen, indem er dichtebasiert arbeitet. Dabei werden zwei zentrale Parameter genutzt: ϵ und $MinPts$. Hierbei definiert ϵ einen Radius um jeden Datenpunkt, während $MinPts$ die Mindestanzahl an Punkten beschreibt, die sich innerhalb des Radius ϵ befinden müssen, damit der jeweilige Punkt als Kernpunkt angesehen wird.

Der DBSCAN-Algorithmus lässt sich in den folgenden Schritten zusammenfassen:

- **Punkte prüfen:** Ist der Punkt schon Teil eines Clusters? Falls nicht, wird versucht, um ihn herum ein neues Cluster zu bilden.
- **Cluster-Bildung:**
  - Es wird gezählt, wie viele Nachbarn sich in einem bestimmten Radius ϵ befinden.
  - Falls es genug Nachbarn ($MinPts$) gibt, wird ein neues Cluster erstellt und der Punkt als Kernpunkt markiert.

  - Alle direkt und indirekt verbundenen Punkte werden dem Cluster hinzugefügt. Dabei können auch Punkte, die selbst nicht die MinPts-Bedingung erfüllen, Teil des Clusters werden, solange sie von einem Kernpunkt aus dichteerreichbar sind.

- **Erweiterung:** Der Prozess wiederholt sich für neu hinzugefügte Punkte, bis keine weiteren dazugehören.
- **Rauschen:** Punkte, die kein Cluster bilden können, werden als Rauschen markiert.
- **Ergebnis:** Alle Punkte sind entweder Teil eines Clusters oder als Rauschen klassifiziert.

Der Algorithmus erkennt Rauschen, also Datenpunkte, die weder Kernpunkte sind noch in der ϵ-Nachbarschaft eines Kernpunktes liegen, also Randpunkte sind. Im Vergleich zu vielen anderen Clustering-Algorithmen benötigt DBSCAN kein Vorwissen über die Anzahl der zu findenden Cluster und ist außerdem in der Lage, auch nicht konvexe Strukturen zu gruppieren.

Der Algorithmus arbeitet effizient und hat eine Laufzeitkomplexität von $O(n \cdot \log n)$ bei n Datenpunkten.

Auch in der Computergrafik findet der Algorithmus seine Anwendung und kann als essentielles Werkzeug zur Analyse von Skizzen dienen. Im Kontext von Landkarten kann DBSCAN also eingesetzt werden, um einzelne Landmassen oder Inseln als eigenständige Einheiten zu erkennen und dementsprechend in nachfolgenden Verarbeitungsschritten zu behandeln.

2.2.2 Alpha Shape

Um α-Shapes verstehen zu können, muss zunächst ein Blick auf die verwandten konvexen Hüllen geworfen werden. Eine konvexe Hülle für eine Punktmenge S beschreibt den Schnitt aller geschlossenen Halbräume, die alle Punkte aus S enthalten. Sie ist somit die kleinste konvexe Menge, die alle Punkte aus S enthält [5].

Eine konvexe Hülle ist allerdings nicht in der Lage, die Form einer Punktmenge genau zu beschreiben und lässt im Extremfall sogar keinerlei Aussagen über diese treffen. Um also die Form genauer beschreiben zu können, können α-Shapes verwendet werden. Ein

α-Shape ist eine Verallgemeinerung der konvexen Hülle, bei der die Form durch die Verwendung von Kreisen mit einem parametrisierbaren Radius beeinflusst wird.

Je nach Wahl dieses Radius wird die Hülle enger an die Punkte angepasst oder bleibt gröber. Mit größerem Radius werden nur die „wesentlichen“ Punkte berücksichtigt, während mit kleinerem Radius die Details der Punktmenge immer feiner erfasst werden. Für $\alpha = 0$ entspricht die α-Shape der konvexen Hülle.

Ein zentraler Aspekt bei der Konstruktion von α-Shapes ist ihr Zusammenhang mit der Delaunay-Triangulation. Je nach Wahl von α basiert die α-Shape entweder auf einer Variante der Delaunay-Triangulation oder der konvexen Hülle. Diese Verbindung ermöglicht eine effiziente Berechnung, da die α-Shape direkt aus der zugrunde liegenden Triangulation abgeleitet werden kann.

Für ein gegebenes α wird geprüft, ob zwei Punkte α-Nachbarn sind. Dafür muss es einen Kreis mit Radius $1/\alpha$ geben, der:

- beide Punkte auf seinem Rand hat,
- alle anderen Punkte der Punktmenge entweder enthält (für $\alpha > 0$) oder nicht enthält (für $\alpha < 0$).

Ein Punkt ist α-extrem, wenn es einen Kreis mit Radius $1/\alpha$ gibt, der:

- den Punkt auf seinem Rand hat,
- alle anderen Punkte der Punktmenge entweder enthält (für $\alpha > 0$) oder nicht enthält (für $\alpha < 0$).

Die α-Shape ist nun der Graph aus:

- allen α-extremen Punkten und den Kanten zwischen Punkten, die α-Nachbarn sind.

α-Shapes können verwendet werden, um aus einer Zeichnung, also einer Menge von Punkten, eine geschlossene Form zu gewinnen. Da die Umrisse von Landmassen auf Landkarten immer geschlossen vorkommen, also zyklisch sind, bietet sich die Nutzung eines solchen Algorithmus an.

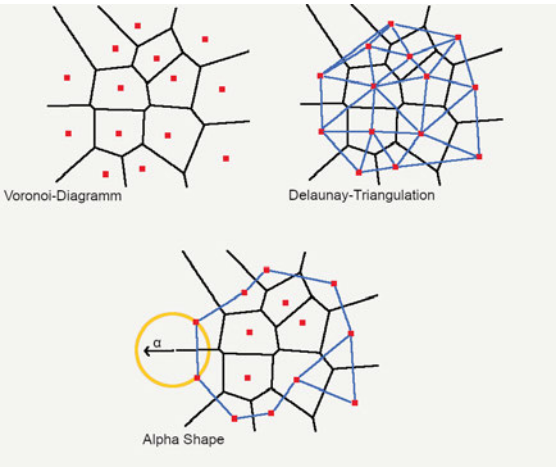

Abbildung 2.3: Voronoi-Diagramm (oben links), Delaunay-Triangulation (oben rechts) und Alpha-Shape (unten) mit Parameter α zur Steuerung des Detailgrads.

[3]

2.3 Prozedurale Content Generierung

Prozedurale Content-Generierung (PCG) bezeichnet die algorithmische Erstellung von Inhalten mit begrenzter oder indirekter Benutzerinteraktion. Sie umfasst die automatisierte Generierung verschiedenster Elemente wie Landschaften, Karten oder Texturen. PCG findet in verschiedenen Bereichen Anwendung, wobei ein häufig genanntes Beispiel die Spieleentwicklung ist. Die Motivation zur Verwendung von PCG ist vielfältig. Neben der Reduktion von Entwicklungszeit und Kosten ermöglicht sie die Schaffung endloser Variationen, adaptiver Spielerfahrungen sowie die Unterstützung von Designern im Entwicklungsprozess. Während PCG bereits seit den 1980er Jahren in kommerziellen Spielen zum Einsatz kommt, gewinnt sie durch neue Forschungsansätze und technologische Möglichkeiten zunehmend an Bedeutung [18]. Die verschiedenen Ansätze zur PCG, die im Rahmen dieser Arbeit betrachtet werden, lassen sich dabei grundsätzlich in regelbasierte und noise-basierte Verfahren unterteilen, die jeweils eigene Stärken und Charakteristika aufweisen.

[3] Eigene Abbildung, nachempfunden basierend auf der Quelle: Zhou, W. und Yan, H. (2012). Alpha shape and Delaunay triangulation in studies of protein-related interactions.

2.3.1 Regel-basiert

Regelbasierte Verfahren sind ein zentraler Ansatz in der PCG. Sie beruhen auf der Definition von klaren Regeln und Constraints, die den Aufbau und die Struktur von Inhalten steuern. Durch die systematische Anwendung dieser Regeln können komplexe und konsistente Strukturen erzeugt werden, die oft sowohl ästhetisch ansprechend als auch logisch kohärent sind. Zu bekannten Vertretern zählen L-Systeme und der Wave Function Collapse Algorithmus, die jeweils unterschiedliche Ansätze verfolgen, um Inhalte prozedural und effizient zu generieren.

L-Systeme

L-Systeme wurden ursprünglich 1968 von dem Biologen Astrid Lindenmayer entwickelt, um das Wachstum von Pflanzen zu modellieren [17]. Sie sind ein formales System zur Beschreibung der Entwicklung von verzweigten Strukturen durch regelbasierte Ersetzungen.

Ein L-System besteht dabei aus folgenden Komponenten:

- dem Alphabet V
- einem Axiom ω
- und Regeln P

Das Alphabet V wird hierbei durch eine Menge von Symbolen repräsentiert, die ersetzt werden können (Variablen) oder nicht ersetzbar sind (Konstanten). Das Axiom ω ist ein Ausgangszustand (Startstring) und besteht aus Symbolen des Alphabets. P ist eine Tabelle von Regeln, wobei der Schlüssel ein bestimmtes Symbol des Alphabets ist und der Wert ebenfalls ein Symbol oder eine Symbolkette ist, durch welches der Schlüssel ersetzt wird.

Ein System kann also anhand der zuvor vorgestellten Werte parametrisiert werden. Ein weiterer Parameter ist die Anzahl der Iterationen, also wie oft der gesamte Regelsatz auf die Zeichenkette angewendet werden soll. Hierbei werden die Regeln in der ersten Iteration nur auf das Axiom angewendet, während die Regeln in der nächsten Iteration auf das Resultat der vorherigen Iteration angewendet werden. Wenn also $x[n]$ den Zustand

nach n Iterationen darstellt mit $x[0] = \omega$, dann ist $x[n+1] = P(x[n])$, wobei $P(x[n])$ die Anwendung der Regeln P auf den Zustand $x[n]$ bedeutet.

Für die grafische Interpretation von L-Systemen wird oft eine sogenannte "Turtle" verwendet. Eine Turtle ist an spezifische Symbole gebunden, wobei d eine variable Länge und δ einen Rotationswinkel bezeichnet. Konventionell ist die Turtle wie folgt definiert:

- F: Zeichnet eine Linie in aktuelle Richtung mit Länge d
- f: Bewegt sich in aktuelle Richtung mit Schrittweite d
- $+$: Rotiert nach links um Winkel δ
- $-$: Rotiert nach rechts um Winkel δ
- [: Speichert aktuellen Zustand der Turtle (Grundlegend die Position und Orientierung)
-]: Stellt den zuletzt gespeicherten Zustand der Turtle wieder her und entfernt diesen aus dem Speicher.

Eine solche Definition reicht aus, um erstaunlich organisch wirkende Strukturen zu modellieren. Besonders für Landkarten gibt es diverse Anwendungsfälle, um L-Systeme dort einsetzen zu können. Ein Beispiel wäre die dynamische Generierung von Flussstrukturen, die auf natürlichen Prozessen wie Sedimentierung und Fließbewegungen basieren. Um Variabilität und Realismus dieser Strukturen zu fördern, können L-Systeme stochastisch erweitert werden, indem unterschiedliche Regeln und Parameter angewendet werden. Diese stochastischen Erweiterungen erlauben es, Flussverzweigungen mit variierenden Winkeln, Längen und Verzweigungsmustern zu erzeugen, wodurch die resultierenden Strukturen natürlicher wirken. Die stochastische Komponente sorgt zudem dafür, dass selbst mit denselben Ausgangsparametern unterschiedliche Flussnetze entstehen können [21].

WFC

Der Wave Function Collapse Algorithmus (WFC), entwickelt von Maxim Gumin, ist eine Methode zur prozeduralen Generierung von Inhalten[4]. Ursprünglich stammt der Begriff aus der Quantenmechanik, wurde jedoch für den von Gumin entwickelten Algorithmus

[4]Maxim Gumin, "Wave Function Collapse", GitHub Repository, https://github.com/mxgmn/WaveFunctionCollapse

zur Generierung von Texturen und Leveldesigns in Computerspielen genutzt. Der Anwendungsbereich entfaltet sich jedoch weiter, sodass eine abgewandelte Form des Algorithmus sogar für die Generierung von Gedichten verwendet wird. Der Algorithmus arbeitet beispielbasiert und erzeugt neue Inhalte, die den lokalen Mustern eines vorgegebenen Beispiels entsprechen [11].

In der ursprünglichen Implementierung existieren zwei Hauptvarianten: Der `Simple Tiled` Ansatz, der mit diskreten Kacheln und festen Nachbarschaftsregeln arbeitet, sowie der `Overlapping` Ansatz, der überlappende Nachbarschaften von Pixeln oder Elementen betrachtet. Beide Varianten folgen denselben Grundprinzipien, unterscheiden sich jedoch in der Definition und Anwendung ihrer Constraints.

Der Algorithmus zerlegt eine Eingabedatei, typischerweise ein Bild, in eine Menge lokaler Muster, die durch überlappende Bereiche kleiner „Unterbilder“ definiert werden. Diese Muster werden als Constraints interpretiert, die die möglichen Anordnungen der Muster in der Ausgabe beschränken.

Im Vergleich zu herkömmlichen Textursyntheseverfahren, bei denen Pixelwerte interpoliert werden, arbeitet der WFC Algorithmus ausschließlich mit diskreten Mustern. Dies macht den Algorithmus besonders geeignet für Anwendungen, bei denen semantische Konsistenz entscheidend ist, wie beispielsweise bei der Generierung von Spielumgebungen.

Der WFC Algorithmus verwendet Techniken zur Lösung von Constraints, wobei jedem Gitterpunkt eine begrenzte Menge an Mustern zugewiesen wird. Durch die Anwendung einer Heuristik wird der Algorithmus dazu gebracht, zunächst die Bereiche mit der größten Einschränkung zu bearbeiten, was zu einem effizienten Algorithmus führt.

Die Arbeitsweise des WFC Algorithmus lässt sich in vier zentrale Schritte unterteilen:

- **Musterextraktion**: Die lokalen Muster werden aus der Eingabedatei extrahiert und katalogisiert.
- **Constraint-Erstellung**: Eine Datenstruktur wird aufgebaut, die die zulässigen Überlappungen der Muster speichert.
- **Iterative Generierung**: Das Ausgabegitter wird schrittweise gefüllt, indem für jeden Punkt ein Muster gewählt wird, das den Constraints entspricht und mit den bereits platzierten Mustern kompatibel ist.

- **Ausgabe**: Das generierte Ergebnis wird als vollständige Konfiguration zurückgegeben.

Die Eigenschaften des Wave Function Collapse Algorithmus eröffnen zahlreiche Möglichkeiten, ihn im Kontext der Generierung von fiktionalen Landkarten einzusetzen. Beispielsweise könnten aus einer groben Skizze detaillierte Hintergrundelemente der Karte generiert werden, indem der WFC Algorithmus lokal konsistente Muster aus vorgegebenen Beispieltexturen anwendet. Die stochastische Natur des Algorithmus fördert dabei die Variabilität.

2.3.2 Noise-basiert

Noise-basierte Verfahren bilden ebenfalls einen fundamentalen Baustein in der PCG. Sie basieren auf der Erzeugung und Manipulation verschiedener Rauschfunktionen, die kontrollierte Zufälligkeit und natürlich wirkende Variationen erzeugen. Durch die geschickte Parametrisierung dieser Funktionen können organische Strukturen und Muster generiert werden, die sowohl visuell überzeugend als auch flexibel anpassbar sind.

Perlin Noise

Perlin Noise ist eine Methode zur Erzeugung von glattem, pseudo-zufälligem Rauschen, die von Ken Perlin eingeführt wurde [14]. Heute ist sie ein grundlegendes Werkzeug in der Computergrafik und wird häufig für prozedurale Textur- und Terraingenerierung verwendet.

Perlin Noise wurde entwickelt, um die Einschränkungen von klassischen, vollständigen Rauschmethoden zu überwinden, die oft zu harten Übergängen und visuellen Artefakten führen. Im Gegensatz dazu weist Perlin Noise die folgenden Eigenschaften auf:

- **Kontinuität:** Perlin Noise erzeugt glatte Übergänge zwischen benachbarten Werten, wodurch organische und natürlich wirkende Muster entstehen.
- **Zufälligkeit:** Die erzeugten Werte wirken zufällig, sind aber deterministisch, sodass sie reproduzierbar sind, wenn der gleiche Startwert verwendet wird.
- **Variabilität:** Durch die Kombination von Perlin Noise auf unterschiedlichen Skalen können komplexere Muster wie Fraktale erzeugt werden [4].

Der ursprüngliche Algorithmus beruht auf einem Punktraster und der Interpolation zwischen den Punkten. Die wichtigsten Schritte sind folgende [14]:

- **Rastererzeugung:** Der Raum wird in ein regelmäßiges Raster unterteilt, wobei jedem Punkt in diesem Raster eine zufällige Gradientenrichtung zugewiesen wird.
- **Skalarprodukt:** Für jeden Punkt im Raum wird das Skalarprodukt zwischen dem Gradienten der Rasterpunkte, in dem der jeweilige Punkt liegt, und dem Vektor der Rasterpunkte zum aktuellen Punkt berechnet. In 2D würden dabei also vier skalare Werte herauskommen.
- **Interpolation:** Die Werte der umliegenden Rasterpunkte werden mithilfe einer glatten Interpolationsfunktion kombiniert. Diese Funktion sorgt für die glatten Übergänge, die für Perlin Noise charakteristisch sind.

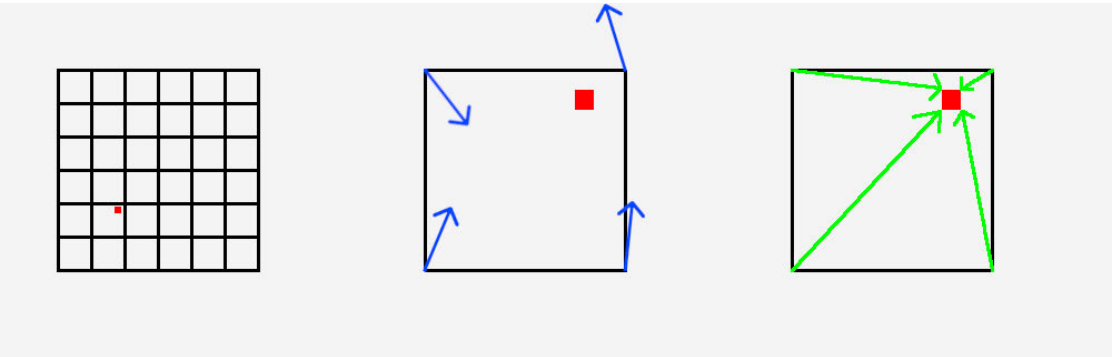

Abbildung 2.4: Schematische Darstellung des Perlin Noise Algorithmus.
Links: Rasterunterteilung des Raums.
Mitte: Ein Punkt innerhalb einer Rasterzelle sowie zufällig zugewiesene Gradienten an den Rasterpunkten.
Rechts: Vektoren von den Rasterpunkten zum betrachteten Punkt.
[5]

Perlin Noise findet zahlreiche Anwendungen in der Computergrafik, insbesondere in der Generierung von natürlich wirkenden Strukturen. Der Algorithmus kann beispielsweise eingesetzt werden, um realistische Texturen für Wasseroberflächen zu erzeugen. Durch die Kombination von Perlin Noise auf verschiedenen Skalen können realistische Landschaften erstellt werden, indem die Noise-Werte als Höhenkarte interpretiert werden.

[5] Eigene Darstellung.

Simplex Noise

Simplex Noise ist eine Weiterentwicklung der Perlin Noise Methode und wurde ebenfalls von Ken Perlin vorgestellt [15]. Diese Methode verbessert vor allem die Performance in höheren Dimensionen und bei größeren Datenmengen. Während die Komplexität von Perlin Noise bei $O(2^n)$ für n Dimensionen liegt, arbeitet der Simplex Noise Algorithmus in $O(n^2)$ [10].

Der Hauptunterschied liegt in der Art, wie das Gitter zur Berechnung des Rauschens aufgebaut ist. Während Perlin Noise ein reguläres, orthogonales Raster nutzt, das aus gleichmäßig verteilten Rasterpunkten besteht, basiert Simplex Noise auf dem sogenannten Simplex-Gitter. Ein Simplex ist die einfachste geometrische Form, die einen n-dimensionalen Raum aufspannen kann. In 2D ist dies beispielsweise ein gleichseitiges Dreieck. Diese Simplexformen haben den Vorteil, dass sie die minimale Anzahl an Eckpunkten für die jeweilige Dimension besitzen. Während ein Quadrat in 2D vier Eckpunkte hat, kommt ein Dreieck mit drei Eckpunkten aus. Dies reduziert die Anzahl der notwendigen Berechnungen erheblich.

Der Algorithmus von Simplex Noise arbeitet in mehreren Schritten [10]:

- **Transformation des Eingaberaums:** Der Raum wird entlang der Hauptdiagonalen verzerrt, sodass die Simplexzellen zu regulären, achsenausgerichteten Hyperwürfeln werden.
- **Bestimmung der Simplexzelle:** Durch Betrachtung der ganzzahligen Koordinatenanteile wird ermittelt, in welcher Zelle sich der zu berechnende Punkt befindet.
- **Traversierung der Rasterpunkte:** Die Rasterpunkte des Simplex werden in einer bestimmten Reihenfolge durchlaufen, die sich aus der Größenordnung der Koordinaten ergibt.
- **Berechnung der Beiträge:** Für jeden Rasterpunkt wird dessen Beitrag zum Rauschen berechnet und aufsummiert, wobei eine Abschwächungsfunktion zum Einsatz kommt.

Diese Methode führt zu effizienterem und qualitativ hochwertigerem Rauschen, das besonders für Anwendungen in höheren Dimensionen geeignet ist, aber ansonsten genau wie Perlin Noise verwendet werden kann.

3 Stand der Technik

In diesem Kapitel werden bestehende Systeme und Methoden vorgestellt, die sich auf die sketch-basierte Generierung von Landschaften und Karten spezialisiert haben. Der Fokus liegt dabei darauf, zu untersuchen, wie diese Ansätze funktionieren, welche Technologien und Algorithmen sie verwenden und wie sie zur Lösung spezifischer Probleme beitragen.

Zunächst wird die Relevanz der sketch-basierten Generierung von Landschaften erläutert, um den praktischen und wissenschaftlichen Wert dieser Techniken zu verdeutlichen. Anschließend werden ausgewählte Projekte und Systeme detailliert beschrieben. Der Vergleich und die Synthese der betrachteten Arbeiten geben schließlich Aufschluss darüber, wie diese Ansätze in den Kontext der vorliegenden Arbeit eingeordnet werden können.

3.1 Relevanz

Die Beschäftigung mit sketch-basierten Methoden in der Generierung von Landschaften ist von großer Bedeutung, da sie eine intuitive und benutzerfreundliche Herangehensweise an ein komplexes Problem bietet, besonders in der Unterhaltungsindustrie, die oft nicht nur realistische, sondern auch stilisierte Darstellungen von Landschaften erfordert.

Ein Vorteil von sketch-basierten Ansätzen liegt in ihrer Fähigkeit, die Komplexität der Generierung zu abstrahieren. Sie bieten dem Nutzer eine direkte Einflussnahme auf die entstehenden Topologien, was bei der Entwicklung von virtuellen Landschaften zugutekommt, da somit eine feingranulare Kontrolle über visuelle Aspekte geschaffen werden kann.

Im Designprozess von Spielen – als Beispiel für viele mögliche Anwendungsgebiete – kann es hilfreich sein, dass die Skizze eines Designers direkt in die virtuelle Welt übertragen werden kann, um darauf aufbauend zu testen und weiterzuentwickeln. Dieser automatisierte Prozess kann somit Zeit und Kosten sparen [19].

3.2 Ansätze und Systeme

3.2.1 Sketch2Map

Eine interessante Herangehensweise wird in der Arbeit Sketch2Map vorgestellt [23], in der das beschriebene System folgendermaßen arbeitet:

Der Nutzer gibt eine handgezeichnete Skizze als Input in das System, welche wichtige Merkmale wie Küstenlinien, Flüsse und andere Geländeformen in einfachen Konturen darstellt. Nun verarbeitet das System die Skizze stufenweise, wobei zuerst ein Conditional Generative Adversarial Network (cGAN) eingesetzt wird, um die Skizze zu einer Art topografischer Karte umzuwandeln. Dabei werden Konturen und Höhenmerkmale basierend auf der Skizze realistisch interpretiert. Anschließend wird die Karte, die Informationen über die Oberflächentopologie des Geländes enthält, als Eingabe für einen deterministischen Algorithmus genutzt. Dieser Algorithmus wandelt die Eingabe in ein konkretes Asset um, welches im weiteren Prozess als 2D- oder 3D-Modell verwendet werden kann.

Da hier Generative Adversarial Networks (GAN) genutzt werden, sind Trainingsdaten erforderlich, die auf unterschiedliche Weise generiert werden:

Einerseits durch prozedural erstellte Welten, bei denen systematisch Höhenkarten erzeugt werden, aus denen sich dann Küstenlinien, Flüsse und andere geografische Merkmale ableiten lassen. Diese Höhenkarten werden durch mehrere Perlin Noise Ebenen generiert. Die so erzeugten Karten werden dann in mehrdimensionale 2D-Bitmap-Darstellungen umgewandelt. Um aus diesen Karten die entsprechenden Trainingsskizzen zu erzeugen, wird eine Interpolation mit verschiedenen Glättungsparametern angewendet, wodurch drei unterschiedliche Stile entstehen. Neben den prozedural generierten Daten werden auch echte Höhendaten der Erde als Trainingsgrundlage verwendet. Mit diesen Trainingspaaren aus Skizzen und entsprechenden Karten wird dann das zweistufige GAN-System trainiert, wobei die erste Stufe die grobe Land-Meer-Segmentierung lernt und die zweite Stufe die detaillierte Höhenkartengeneration innerhalb dieser Segmente übernimmt.

3.2.2 Mapgen4

Eine weitere interessante Herangehensweise findet sich im Projekt Mapgen4 von Amit Patel, das für die interaktive Erstellung von Landkarten entwickelt wurde[1]. Dieses System ermöglicht es dem Nutzer, durch Skizzen direkt Einfluss auf die Gestaltung der Karte zu nehmen. Dabei können gezielt Landschaftsformen wie Berge, Täler oder Gewässer an den gewünschten Stellen gezeichnet werden. Das Besondere ist, dass das System automatisch Flüsse generiert und Biome auf Grundlage einer physikalischen Simulation von Faktoren wie Wind, Verdunstung und Niederschlag berechnet. Das System arbeitet hauptsächlich prozedural. Der zugrunde liegende Algorithmus basiert auf einer Delaunay-Triangulation und ihrer dualen Voronoi-Diagramm-Struktur, die eine effiziente Verarbeitung und Darstellung geografischer Merkmale ermöglicht. Die erzeugten Karten werden in Echtzeit gerendert, was dem Nutzer ein unmittelbares visuelles Feedback bietet und iteratives Arbeiten erleichtert.

Ein zentraler Bestandteil ist die Berechnung der Höhenwerte für die Landkarte[2]. Hierbei greift das Projekt auf unterschiedliche Methoden zurück und kombiniert diese. Dafür wird Simplex Noise verwendet, um grobe Landmassen und Wasserstrukturen vorab aus einem Startwert zu generieren. Durch diese Technik entstehen natürlich wirkende Küstenlinien, die als Basis für die weitere Bearbeitung dienen. Die Veränderungen durch den Nutzer werden vom System erneut durch Simplex Noise verfeinert, sodass die Übergänge zwischen Land und Wasser natürlicher wirken und kleinräumige Details wie Buchten oder Küstenlinien hinzugefügt werden. Gebirgsketten werden ebenfalls durch Simplex Noise generiert, das die Verteilung und Ausrichtung dieser bestimmt. Die Verfeinerung geschieht allerdings durch die Einbeziehung von sogenannten Distanzfeldern, die ebenfalls als Grundlage für die Höhenverteilung dienen. Distanzfelder messen den Abstand von jedem Punkt auf der Karte zu bestimmten Merkmalen, wie beispielsweise den Abstand zur Küstenlinie. Diese Werte werden dann genutzt, um die Höhe der Punkte realistisch zu interpolieren. Für die Verteilung von individuellen Bergen wird ebenfalls eine spezielle Art von Noise, nämlich “Blue-Noise”, verwendet, was dafür sorgt, dass die Berge relativ gleichmäßig, aber nicht regelmäßig angeordnet sind [20].

Der Renderer arbeitet auf einer Delaunay-Triangulation, wobei jedes Dreieck mit einem der vier Typen assoziiert werden kann: Ozean, Flussquelle, Flussbiegung, Flussgabelung. Je nach Typ wird eine unterschiedliche Strömung simuliert und eine davon abhängige

[1] Amit Patel, "Mapgen4", https://simblob.blogspot.com/search/label/mapgen4
[2] Amit Patel, "Mapgen4: elevation", https://simblob.blogspot.com/2018/08/mapgen4-elevation.html

Textur gewählt[3]. Um die Flüsse an den Biegungen nicht zu kantig zu gestalten, da das Eintreten des Flusses auf einer Seite des Dreiecks und das Austreten auf einer anderen Seite mit einer trivialen Methodik dazu führen könnte, werden Bézier-Kurven genutzt. Zur Darstellung des Flusssystems wird ein binärer Baum konstruiert, wobei für jeden Fluss ein Baum entsteht. Von der Küstenlinie ausgehend werden die Flüsse stromaufwärts aufgebaut. Regen wird simuliert, indem das Wasser von den Blättern des aufgebauten Baumes stromabwärts, also zu den Elternknoten, fließt. Während dieses Prozesses wird für jeden Knoten der Niederschlag berechnet, die Flussmenge des aktuellen Knotens um den Regen erhöht und die Flussmenge dem Elternknoten hinzugefügt.

Anstatt die herkömmlichen Perspektiven wie eine reine Draufsicht oder Seitenansicht zu verwenden, werden hier diese beiden Methoden zu einer einzigen Darstellung kombiniert. Dadurch entstehen Karten, bei denen Eigenschaften der Draufsicht wie Flüsse und Küstenlinien zusammen mit Seitenansichten von Bergen dargestellt werden. Dafür werden die Informationen, die in der z-Koordinate der Punkte enthalten sind, in die y-Koordinate des jeweiligen Punktes übertragen. Wenn also Informationen in der z-Koordinate enthalten sind, wie es durch die Höhenkarte bei Bergen der Fall ist, dann wird diese Information durch eine Abbildung auf die y-Achse übertragen, also „nach oben“.

Durch eine Kombination aus geometrischen Algorithmen und physikalischen Simulationen stellt Mapgen4 ein leistungsstarkes Werkzeug für die prozedurale Kartenerstellung bereit. Es bietet eine wertvolle Perspektive auf Ansätze, die auch für diese Arbeit relevant sein können.

3.2.3 Terrain Sketching

In der Arbeit Terrain Sketching wird ebenfalls ein sketch-basierter Ansatz präsentiert, der in der Lage ist, Gelände zu generieren [8]. Das System ermöglicht es Nutzern, Landschaftsformen intuitiv durch Striche zu definieren, die dann anschließend in dreidimensionales Terrain umgewandelt werden. Das System arbeitet mit drei verschiedenen Interaktionsmodi:

Im Silhouetten-Modus zeichnen Nutzer Höhenprofile von Landschaftsformen, die auf eine vertikale Ebene projiziert werden. Der Aerial-Modus ermöglicht das Zeichnen aus der Vogelperspektive, was besonders für einschneidende Landschaftsformen wie Canyons

[3] Amit Patel, "Mapgen4: river appearance", https://simblob.blogspot.com/2018/09/mapgen4-river-appearance.html

geeignet ist. Im Region-Modus können Gebiete markiert werden, um deren Oberflächenbeschaffenheit zu modifizieren. Ein zentraler Aspekt des Systems ist die Verarbeitung der Benutzerstriche. Aus einer gezeichneten Silhouette wird automatisch eine Schattenkurve durch Schnitt mit dem existierenden Terrain erzeugt. Zusätzlich wird eine Begrenzungskurve generiert, die die seitliche Ausdehnung der Landschaftsform definiert. Diese wird basierend auf den "Schulterregionen" der Silhouette berechnet, also den Bereichen, die zu lokalen Maxima oder Minima führen. Die Form dieser Begrenzungskurve passt sich automatisch an. Bei einzelnen Gipfeln wird sie kreisförmig, bei Bergketten länglich.

Die finale Generierung basiert auf einer Oberflächendeformation auf mehreren Auflösungsebenen. Für jede Auflösungsebene wird das Gelände innerhalb der Begrenzungskurve durch eine Kombination aus "Wavelet Noise" und Deformation angepasst. Die Varianz des Rauschens wird dabei aus der Analyse der Silhouettenkurve abgeleitet. Die Deformation selbst erfolgt durch eine kurvenbasierte räumliche Verformung.

Durch diese Kombination aus intuitiver sketch-basierter Eingabe und komplexer algorithmischer Verarbeitung ermöglicht das vorgestellte System die effiziente Erstellung realistischer Landschaftsformen, wobei hier ein ganz klarer Fokus auf die Darstellung von Höhenmerkmalen gelegt wird.

3.3 Vergleich und Synthese

Die analysierten Systeme demonstrieren unterschiedliche Herangehensweisen an die sketchbasierte Generierung von Landschaften, die sowohl auf algorithmischer Komplexität als auch auf Nutzerfreundlichkeit abzielen. Trotz ihrer Unterschiede lassen sich Gemeinsamkeiten sowie komplementäre Ansätze identifizieren, die wichtige Erkenntnisse für eine eigene Implementierung bieten.

Alle betrachteten Systeme verfolgen das Ziel, dem Nutzer eine intuitive und interaktive Möglichkeit zu bieten, Landschaften durch einfache Skizzen oder Eingaben zu erstellen. Dabei werden grundlegende Geometrien und topografische Merkmale wie Berge, Flüsse und Küstenlinien aus den Skizzen extrahiert und durch algorithmische Prozesse verfeinert.

Die Systeme setzen außerdem auf verschiedene Methoden zur Höhenerzeugung und zur Strukturierung geografischer Merkmale. Sowohl Sketch2Map als auch Mapgen4 und Ter-

rain Sketching nutzen Noise-Algorithmen, um natürliche Übergänge und realistische Details zu generieren.

Die Unterschiede der Systeme liegen vor allem in den eingesetzten Algorithmen und der Zielsetzung der jeweiligen Ansätze. Sketch2Map setzt auf ein zweistufiges GAN-System, das insbesondere für eine variable Stilisierung des Ergebnisses geeignet ist, während Mapgen4 und Terrain Sketching mit einer stärker prozeduralen Herangehensweise arbeiten, die physikalische Simulationen und geometrische Algorithmen kombiniert. Dieser Fokus macht Mapgen4 und Terrain Sketching zu interaktiven Werkzeugen, die besonders für kreative Anwendungen wie die Erstellung von Karten in Spielen oder für *"Storytelling"* geeignet sind.

Terrain Sketching bietet durch seine drei Interaktionsmodi eine hohe Flexibilität in der sketch-basierten Benutzereingabe und legt dabei besonderen Wert auf die intuitive Definition von Höhenmerkmalen. Hier steht die detaillierte Verarbeitung einzelner Striche im Vordergrund, wodurch Nutzern eine präzise Kontrolle über die erzeugten Landschaftsformen ermöglicht wird.

Die Stärken der analysierten Ansätze bieten wertvolle Hinweise für eine eigene Implementierung. Insbesondere die Integration von physikalischen Simulationen und geometrischen Algorithmen könnte helfen, realistische und anpassbare Landschaften zu generieren. Gleichzeitig bietet die GAN-basierte Methode von Sketch2Map Potenzial, um realistische topografische Details automatisch aus groben Skizzen abzuleiten.

Die Synthese dieser Ansätze könnte ein System hervorbringen, das sowohl künstlerischen als auch funktionalen Anforderungen gerecht wird. Ein solches System könnte eine flexible Skizzeneingabe mit prozeduralen und datenbasierten Techniken kombinieren, um sowohl stilisierte als auch realistische Landschaften zu erstellen. Besondere Aufmerksamkeit sollte dabei auf die Benutzerfreundlichkeit und das unmittelbare visuelle Feedback während der Eingabe gelegt werden, um iterative Designprozesse zu fördern.

4 Konzept

Dieses Kapitel bildet die Brücke zwischen der bisher behandelten theoretischen Basis und einer konkreten Lösungsimplementierung der einleitend skizzierten Herausforderungen. Es beschreibt den zentralen Ansatz zur Umsetzung der sketch-basierten Generierung von fiktionalen Landkarten. Ziel ist es, die funktionalen und nicht-funktionalen Anforderungen an das System klar zu definieren und ein methodisches Vorgehen zu entwickeln, das die Grundlagen für die praktische Implementierung bildet.

Im Folgenden werden zunächst die gewünschten Funktionen und Eigenschaften des Systems beschrieben, bevor auf übergeordnete Qualitätsmerkmale eingegangen wird. Darauf aufbauend wird der grundlegende Lösungsansatz skizziert, der beschreibt, wie die einzelnen Verarbeitungsschritte von der Skizzeneingabe bis zur fertigen Karte ineinandergreifen. Abschließend wird das methodische Vorgehen erläutert, das den iterativen Entwicklungsprozess der Umsetzung beschreibt.

4.1 Funktionale Anforderungen

Im Folgenden wird darauf eingegangen, was das System mindestens leisten soll, und es werden die funktionalen Anforderungen spezifiziert. Die Anforderungen orientieren sich an der erwarteten Interaktion des Nutzers mit dem System und bilden eine wichtige Grundlage für die Systemarchitektur und Implementierung.

4.1.1 Sketch-Eingabe und Segmentierung der Regionen

Ein zentraler Bestandteil des Systems ist die intuitive Sketch-Eingabe durch den Nutzer. Der Nutzer hat hierbei die Möglichkeit, auf einer Zeichenfläche mit der Maus beliebig und freihand zu zeichnen und gegebenenfalls Korrekturen vorzunehmen. In diesem Schritt des Prozesses geht es also nur um den Umriss einer Landkarte. Ist der Nutzer zufrieden mit

dem Ergebnis seiner Skizze, dann kann dieser die Sketch-Eingabe beenden. Diese Skizze dient als Grundlage für die weitere Verarbeitung und stellt einen Ausgangspunkt für die Generierung der Karte dar. Es ist essentiell, dass das System auch ungenaue Eingaben zuverlässig interpretiert. Da eine Landkarte aus einem oder mehreren geschlossenen Umrissen besteht, müssen gezeichnete Konturen zu vollständigen Polygonen verarbeitet werden. Das System ist also auch in der Lage, mit Insel-artigen Umrissen umzugehen und kann somit mehrere Umrisse gleichzeitig behandeln. Somit generiert das System geschlossene Formen, die den ursprünglichen Umriss bestmöglich approximieren und gleichzeitig die gewünschte natürliche Ästhetik beibehalten.

Der Nutzer wird, nachdem die Skizze geschlossen wurde, die Umrisse der Landkarte in Regionen unterteilen können. Hierfür ist ein Icon-Platzierungs-Mechanismus vorgesehen. Der Nutzer kann dabei zwischen unterschiedlichen Icons wie dem Berg-Icon, Wald-Icon, See-Icon oder Dorf-Icon wählen und diese beliebig auf der Zeichenfläche platzieren. Das System prüft daraufhin, ob sich die platzierten Icons innerhalb eines Umrisses befinden oder ob sie ignoriert werden. Die validen Icons bilden dann zusammen mit dem geschlossenen Umriss die Grundlage für die Unterteilung in Regionen beziehungsweise geografische Zonen, welche jeweils mit einer Landschaftsart assoziiert werden können.

Die konkreten Anforderungen an das System für diesen Schritt sind also:

- Zeichnungen können Pixelgenau eingelesen werden.
- Zeichnungen sind korrigierbar.
- Linien mit Unterbrechungen von bis zu 5 Pixeln werden automatisch verbunden und unvollständig geschlossene Formen werden dabei erkannt.
- Details von Sketch-Eingaben bleiben auch nach der Verarbeitung erhalten. Dabei wird durch Stichproben an mindestens 20 charakteristischen Punkten (wie Ecken, Einbuchtungen und Ausbuchtungen) überprüft, ob die Pixel höchstens um 5 Pixel von ihrer ursprünglichen Position abweichen.
- Inselstrukturen werden korrekt, also individuell, behandelt.
- Icons werden validiert.
- Icons unterteilen die Fläche in Teilmengen, die mit Landschaftstypen assoziiert werden.

4.1.2 Landschafts Generierung

Nach der Vorverarbeitung der Landkarte in geografische Zonen ist der nächste Schritt die Generierung der Landschaften, die den definierten Regionen zugewiesen werden. Dieser Prozess stellt den Kern der Anwendung dar und sorgt dafür, dass die Landkarte mit Landschaftsdetails versehen wird, die zusammen eine typische Darstellung von Fantasy-Karten ergeben. Diese Karten zeichnen sich durch stilisierte, aber dennoch plausible geografische Merkmale aus, die in fiktionalen und oft magischen Welten Verwendung finden. Zu diesen Merkmalen gehören beispielsweise Gebirge, Wälder, Seen, Flüsse und Dörfer. Diese Elemente sind klar voneinander abgegrenzt, gehen jedoch in ihren Übergängen harmonisch ineinander über.

Die Landschaftsmodellierung zielt darauf ab, eine Karte zu erzeugen, die sowohl den spezifischen visuellen Anforderungen von Fantasy-Karten entspricht, als auch geografisch konsistent bleibt. Übergänge von Gebirgsregionen zu Wäldern müssen beispielsweise sanft und organisch wirken, um ein glaubhaftes visuelles Erlebnis zu schaffen. Die generierten Landschaften sollen nicht nur die ästhetischen Merkmale einer Fantasy-Karte widerspiegeln, sondern auch eine gewisse Authentizität und Kohärenz in den geografischen Beziehungen der Zonen bieten.

Ein weiteres Ziel ist die Einzigartigkeit der generierten Karten: Auch bei gleichen Eingaben sollen leicht abweichende Ergebnisse erzielt werden, um die Individualität der Karten zu gewährleisten.

Zusammengefasst ergeben sich die folgenden Anforderungen:

- Stilisierte Landschaften werden auf Basis der jeweiligen Icons der Regionen generiert.
- Übergänge zwischen Landschaften werden so gestaltet, dass sie realistisch und fließend wirken.
- Die Karte wird mit bekannten Details ausgestattet, wie organisch wirkenden Flussverläufen, Pergament-Optik und Namen für Dörfer.
- Die Generierung liefert einzigartige Ergebnisse, selbst bei identischen Eingaben.

4.1.3 Ausgabe

Nachdem die Landkarte erfolgreich generiert wurde, muss das System das Resultat als Bild ausgeben. Der Fokus liegt dabei auf der Bereitstellung einer einfachen und effizienten Möglichkeit, die fertige Landkarte als Bilddatei zu speichern. Der Export als Bild ermöglicht es den Nutzern, ihre generierte Karte in einem gängigen Format zu speichern und weiterzuverwenden.

Das System muss die erzeugte Landkarte als Bild rendern. Dabei wird der gesamte Kartenausschnitt, einschließlich der generierten Landschaftstypen, in einem Bild zusammengeführt. Die generierte Karte sollte alle visuellen Elemente wie Farben, Texturen und Übergänge berücksichtigen, um das gewünschte ästhetische Ergebnis zu liefern. Für die Erstellung des Bildes wird ein Rendering-Mechanismus eingesetzt, der die einzelnen Kartenelemente basierend auf den festgelegten Parametern kombiniert und in ein pixelbasiertes Format überführt.

Der Nutzer kann die Landkarte direkt nach der Generierung in ein übliches Bildformat exportieren. Eine benutzerfreundliche Oberfläche wird bereitgestellt, die es ermöglicht, die Bilddatei mit einem Klick zu speichern.

Zusammengefasst ergeben sich die folgenden Anforderungen:

- Die gesamten zuvor generierten Informationen werden in einem einzigen Bild gerendert.
- Das Bild ist beispielsweise als `png` oder `jpg` exportierbar.

4.2 Nicht-funktionale Anforderungen

Neben den funktionalen Anforderungen spielen auch die nicht-funktionalen Anforderungen eine wesentliche Rolle bei der Entwicklung des Systems. Diese Anforderungen definieren qualitative Eigenschaften des Systems, die für die Nutzererfahrung und die langfristige Nutzbarkeit entscheidend sind. Sie betreffen Aspekte wie Effizienz, Benutzerfreundlichkeit und Anpassungsfähigkeit. Folgende nicht-funktionale Anforderungen sind essentiell, um sicherzustellen, dass das System den gewünschten Qualitätsstandards entspricht und langfristig genutzt werden kann.

4.2.1 Performance

Die Performance des Systems ist ein entscheidender Faktor für eine positive Nutzererfahrung. Insbesondere bei der Generierung und Darstellung komplexer Karten muss das System schnell und effizient arbeiten. Die Berechnung der Landkarte, einschließlich der Verarbeitung der Sketch-Eingabe und der Landschaftszuordnung, sollte innerhalb eines angemessenen Zeitrahmens erfolgen, um eine flüssige Interaktion zu ermöglichen. Auch der Export der Landkarte als Bild muss zügig ablaufen, damit Nutzer nicht unnötig warten müssen. In Bezug auf die Performance bedeutet dies, dass das System die Eingaben der Nutzer innerhalb von wenigen Sekunden verarbeiten und die fertige Karte erstellen können sollte.

Hierbei gibt es jedoch eine Ausnahme: Das Modul, das für das Einlesen der Zeichnung zuständig ist, muss die Ergebnisse in Echtzeit anzeigen, so wie man es von üblichen Zeichenprogrammen kennt. In dieser Arbeit bezieht sich der Begriff Echtzeit auf eine Nutzerinteraktion ohne wahrnehmbare Verzögerung. Dies bedeutet, dass das System so schnell auf Benutzereingaben reagiert, dass die Reaktionszeit für den Menschen als unmittelbar empfunden wird. Für alle weiteren Schritte, wie der Approximierung des Umrisses, wird eine gewisse Berechnungszeit eingeplant.

Durch die Sicherstellung einer guten Performance wird gewährleistet, dass Nutzer das System effektiv und ohne Frustration verwenden können.

Das System garantiert:

- Eine Echtzeitanzeige der gezeichneten Skizze.
- Der Berechnungsschritt, der für die Segmentierung der Regionen sorgt, braucht maximal eine Sekunde.
- Die Landschaftsgenerierung erfolgt in weniger als fünf Sekunden.
- Keine Überlastung des Systems durch Karten mit bis zu 20 Icons, sodass alle Anforderungen weiterhin erfüllt werden.

4.2.2 Erweiterbarkeit

Die Erweiterbarkeit des Systems ist von entscheidender Bedeutung, um sicherzustellen, dass es auch in Zukunft an neue Anforderungen oder Nutzungsgewohnheiten angepasst und erweitert werden kann. Das System wird so entwickelt, dass mit minimalem Aufwand neue Funktionen oder Komponenten integriert werden können.

Die Erweiterbarkeit des Systems wird durch folgende Merkmale gewährleistet:

- Der modulare Aufbau ermöglicht eine einfache Integration neuer Funktionen, da jede Komponente klar abgegrenzt ist.
- Jedes Modul erfüllt eine spezifische Aufgabe, die in sich abgeschlossen ist, was die Wartbarkeit erhöht.
- Eine geringe Kopplung erleichtert die Austauschbarkeit von Komponenten und minimiert unerwartete Seiteneffekte an anderen Stellen im System.
- Abstraktionen und Schnittstellen machen das System flexibler, da Anpassungen vorgenommen werden können, ohne die interne Logik im Detail ändern zu müssen.
- Eine Steuerung durch Parameter ermöglicht Anpassungen am System, ohne tief in den Quellcode eingreifen zu müssen.
- Gut dokumentierter Code sorgt dafür, dass das System auch bei zukünftigen Weiterentwicklungen leicht verständlich bleibt.

Durch die Schaffung einer robusten und erweiterbaren Architektur wird das System zukunftssicher gestaltet und bleibt in der Lage, flexibel auf neue Anforderungen zu reagieren.

4.3 Von der Skizze zur Karte

Zur systematischen Transformation von Skizzen in detaillierte Landkarten wird eine mehrstufige Verarbeitungspipeline implementiert, die folgende Schlüsselkomponenten umfasst:

1. **Skizzenverarbeitung**

- **Bresenham-Algorithmus**: Gewährleistet ein natürliches Gefühl beim Zeichnen der Umrisse.
- **DBSCAN-Clustering**: Gruppiert die Pixel und stellt sicher, dass mehrere Landmassen korrekt gezeichnet werden können.
- **Delaunay-Triangulation**: Dient als Grundlage für die Berechnung der Alpha Shapes und der Voronoi-Diagramme.
- **Alpha Shape**: Schließt mögliche Lücken in den Umrissen und korrigiert Fehler in der Skizze.

2. **Regionsegmentierung**
   - **Voronoi-Diagramm**: Unterteilt die Landmasse basierend auf platzierten Icons in Zellen.
3. **Regiongenerierung**
   - **Perlin Noise und Simplex Noise**: Erzeugen organische Verteilungen für Berge und Ozeane.
   - **Wave Function Collapse (WFC)**: Generiert strukturierte Wälder, die aus einzelnen Bäumen bestehen.
4. **Integration**
   - **L-System**: Erzeugt organische Flussstrukturen über die gesamte Karte hinweg.
   - **Filter**: Harmonisieren die Übergänge und fügen Texturen hinzu.

Die einzelnen Phasen bauen aufeinander auf. Die Ausgabe eines Schrittes dient dabei als Eingabe für den nächsten.

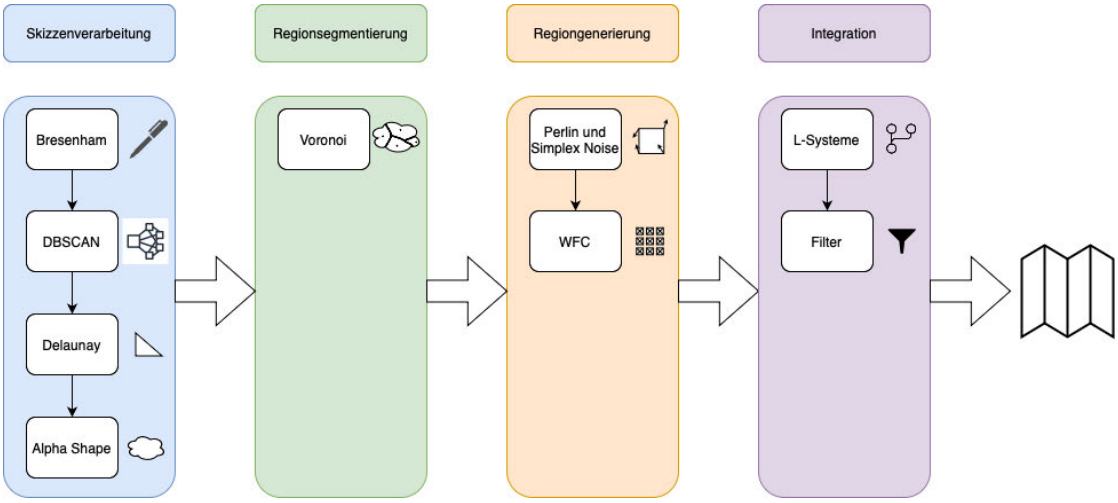

Abbildung 4.1: Pipeline unterteilt in die Phasen Skizzenverarbeitung, Regionsegmentierung, Regiongenerierung und Integration.

4.4 Methodisches Vorgehen

Die Entwicklung des Systems folgt einem agilen, iterativen und komponentenbasierten Ansatz, bei dem zunächst grundlegende Funktionalitäten implementiert und dann schrittweise erweitert werden. Das Vorgehen gliedert sich in mehrere aufeinander aufbauende Phasen.

Die erste Phase widmet sich der Entwicklung des grundlegenden Applikationsgerüsts mit Fokus auf die Sketch-Eingabe. Es wird ein Framework geschaffen, das Basisinteraktionen ermöglicht und eine robuste Routing-Struktur zwischen verschiedenen Anwendungszuständen bereitstellt. Die zentrale Herausforderung besteht in der Implementierung einer interaktiven Zeichenfläche, die präzise Benutzereingaben erfassen kann.

In der zweiten Phase liegt der Fokus auf der Analyse und Optimierung der Sketch-Eingabe. Es werden Methoden entwickelt, die die Freihandzeichnung des Nutzers analysieren und in eine strukturierte Form überführen. Dies umfasst die Identifikation und Bereinigung von Ungenauigkeiten sowie die Generierung geschlossener Polygone.

Die dritte Phase behandelt die Icon-basierte Segmentierung der Karte. Auf Basis des generierten Polygons wird ein System entwickelt, das die Platzierung von Icons ermöglicht

und diese zur Unterteilung der Fläche in distinkte Regionen nutzt. Die Entwicklung effizienter Algorithmen zur räumlichen Analyse spielt hierbei eine zentrale Rolle.

Die vierte Phase konzentriert sich auf die prozedurale Generierung der verschiedenen Landschaftstypen. Für jeden Landschaftstyp werden spezifische Generierungsstrategien entwickelt, die charakteristische Merkmale der jeweiligen Landschaft erzeugen. Aufgrund der modularen Struktur können diese unabhängig voneinander betrachtet werden.

In der abschließenden Phase geht es um die Zusammenführung der Informationen zu einem Gesamtbild, sowie die Ergänzung kleiner Details. Besondere Aufmerksamkeit gilt dabei der Entwicklung von Methoden für überzeugende Übergänge zwischen den verschiedenen Landschaftstypen.

Die Phasen bauen logisch aufeinander auf, wobei jede Phase die Grundlage für die nachfolgende bildet. Dadurch können grundlegende Funktionalitäten frühzeitig getestet und schrittweise erweitert werden, während die Gesamtarchitektur des Systems konsistent bleibt. Dies wird vor allem durch die erste Phase ermöglicht, da hier durch den Aufbau einer modularen Anwendungsstruktur mit klar definierten Verantwortlichkeiten die Grundlage für erweiterbare und wartbare Komponenten geschaffen wird. Die gewählte Architektur erlaubt es, neue Funktionalitäten schrittweise zu integrieren, ohne bestehende Komponenten grundlegend überarbeiten zu müssen.

Die beschriebenen Phasen können im Kontext agiler Entwicklung als Epics betrachtet werden, die sich in konkrete User Stories und Tasks untergliedern lassen. Dieser Ansatz ermöglicht eine flexible und iterative Entwicklung, bei der einzelne Funktionalitäten schrittweise implementiert und verfeinert werden können. So kann beispielsweise die Sketch-Eingabe zunächst mit grundlegenden Zeichenfunktionen umgesetzt und in späteren Iterationen um Features wie Korrekturmöglichkeiten erweitert werden. Diese agile Herangehensweise erlaubt es, frühzeitig Feedback zu einzelnen Komponenten einzuholen und potenzielle Probleme rechtzeitig zu erkennen und zu beheben, während die übergeordnete Struktur der Entwicklungsphasen als Orientierungsrahmen bestehen bleibt.

5 Umsetzung

Die Umsetzung der sketch-basierten Generierung fiktionaler Landkarten erforderte einen ganzheitlichen Ansatz, der theoretische Konzepte, softwaretechnische Lösungen und kreative Algorithmen vereint. In den folgenden Abschnitten wird der Weg von der initialen Idee bis zur funktionsfähigen Anwendung detailliert beschrieben. Der Fokus liegt dabei auf den technologischen Entscheidungen, der Architektur des Systems und den spezifischen Implementierungsstrategien, die entwickelt wurden, um aus einer einfachen Skizze eine komplexe und lebendige Landkarte zu generieren.

5.1 Technologie

Die praktische Umsetzung basiert auf einem speziell zusammengestellten Technologiestack, der die zentralen Anforderungen an die prozedurale Generierung und die Verarbeitung benutzerdefinierter Skizzen adressiert. Die verwendeten Technologien lassen sich in Programmiersprachen, Bibliotheken und Werkzeuge gliedern:

Programmiersprache

Die wichtigste Entscheidung ist hierbei wahrscheinlich die Wahl der Programmiersprache, da alle anderen genannten Technologien davon abhängig sind.

Java mit JDK 17[1] : Java dient als Kerntechnologie für die Implementierung der Anwendung. Die Wahl fiel auf Java aufgrund seiner Plattformunabhängigkeit und seiner umfangreichen Bibliotheken. Diese spielen insbesondere bei der Verarbeitung geometrischer Strukturen und bei rechenintensiven Algorithmen für die prozedurale Generierung eine zentrale Rolle.

[1] Java, https://docs.oracle.com/en/java/javase/17/

Frameworks und Bibliotheken

JavaFX[2] : Für die Benutzerinteraktion und Visualisierung kommt JavaFX zum Einsatz. Es ermöglicht eine intuitive und interaktive Benutzeroberfläche, die für die Eingabe und Bearbeitung von Skizzen optimiert ist. Darüber hinaus wird JavaFX genutzt, um die generierten fiktionalen Landkarten in einer ansprechenden grafischen Darstellung auszugeben. JavaFX erleichtert außerdem die Umsetzung eines Model-View-Controller-Paradigmas durch Eigenschaften wie die Trennung von FXML-Dateien zur Gestaltung der Benutzeroberfläche und die Verwendung von Java-Klassen für die Steuerung der Views und Models.

Java Topology Suite (JTS)[3] : Diese Bibliothek ist essenziell für die geometrische Verarbeitung und Analyse der eingegebenen Skizzen. Sie stellt Funktionen zur Verfügung, um komplexe Algorithmen wie Delaunay-Triangulationen, Voronoi-Diagramme oder die Erkennung geometrischer Formen effizient zu implementieren. JTS ermöglicht es, aus den skizzierten Eingaben präzise Regionen und Grenzen abzuleiten.

Auburn FastNoiseLite[4] : Für die prozedurale Generierung wird Auburn FastNoiseLite verwendet. Die Bibliothek bietet eine schnelle und vielseitige Implementierung von Noise-Algorithmen wie Perlin oder Simplex Noise, die zur Generierung von realistisch wirkenden Landschaftsmerkmalen wie Höhenkarten oder Gebirgszügen genutzt werden.

Allison Casey WaveFunctionCollapse[5] : Diese Java-Adaption des WFC Algorithmus wird genutzt, um strukturierte und kohärente Kartenbereiche aus vorgegebenen Mustern zu generieren.

Apache Commons[6] : Diese vielseitige Bibliothek unterstützt die Entwicklung durch grundlegende Hilfsfunktionen, etwa für mathematische Operationen oder Datenverarbeitung.

[2] JavaFX, https://openjfx.io/
[3] Java Topology Suite (JTS), https://github.com/locationtech/jts
[4] Auburn FastNoiseLite, https://github.com/Auburn/FastNoiseLite
[5] Allison Casey WaveFunctionCollapse, https://github.com/allison-casey/wavefunctioncollapse
[6] Apache Commons, https://commons.apache.org/

Werkzeuge

IntelliJ IDEA[7] : Die Entwicklung der Anwendung erfolgt in IntelliJ IDEA, einer leistungsstarken Entwicklungsumgebung, die speziell für große, komplexe Projekte geeignet ist. Sie bietet Funktionen wie automatische Codeanalyse, Debugging-Tools und Codesuche, wodurch die Effizienz bei der Implementierung erheblich gesteigert wird.

Gradle[8] : Gradle dient als Build-Tool und unterstützt die Automatisierung von Build-Prozessen, die Verwaltung von Abhängigkeiten und die Integration der eingesetzten Bibliotheken.

Git[9] : Für die Versionskontrolle wird Git verwendet. Es ermöglicht die Nachverfolgung von Änderungen während der Entwicklung und bietet Sicherheit bei der Arbeit an verschiedenen Aspekten der Anwendung.

5.2 Systemarchitektur

Die Systemarchitektur bildet das Grundgerüst der Anwendung und definiert, wie die einzelnen Komponenten miteinander interagieren und welche Entwurfsprinzipien bei der Entwicklung verfolgt wurden. Ziel war es, eine modulare, erweiterbare und wartbare Softwarearchitektur zu schaffen, die die Anforderungen der sketch-basierten Generierung fiktionaler Landkarten effizient abbilden kann. Dabei orientierte sich die Architektur an bewährten Entwurfsmustern, wie sie von Gamma et al. [9] beschrieben sind, und nutzt diese zur Gestaltung einer flexiblen und skalierbaren Lösung. Die nachfolgenden Abschnitte erläutern die technischen und fachlichen Bausteine sowie zentrale Entwurfsmuster, die bei der Implementierung zum Einsatz kamen.

5.2.1 Technische Bausteine

Die technische Architektur des Systems basiert auf einem modularen und flexiblen Design, das auf den Prinzipien von Skalierbarkeit, Erweiterbarkeit und einer klaren Trennung der Verantwortlichkeiten aufgebaut ist. Das System folgt einer modifizierten Model-

[7] IntelliJ IDEA, https://www.jetbrains.com/idea/
[8] Gradle, https://gradle.org/
[9] Git, https://git-scm.com/

View-Controller-Architektur (MVC), um die Kernlogik, die Benutzeroberfläche und die Steuerung der Anwendung klar voneinander zu trennen.

Die Anwendung gliedert sich in folgende technische Hauptkomponenten:

- **Controller**: Verarbeitet Benutzerinteraktionen und koordiniert die Daten zwischen Model und View.
- **View**: Stellt die Benutzerschnittstelle dar und visualisiert die vom Controller gelieferten Daten, während sie Benutzereingaben erfasst und an den Controller weitergibt.
- **Model**: Repräsentiert die Daten und enthält nur simple Geschäftslogik. Daher kann das Model auch als sogenanntes Data Transfer Object (DTO) betrachtet werden.
- **Service**: Kapselt komplexere Geschäftslogik, die auf Models angewendet wird.
- **Util**: Bietet eigenständige Hilfsfunktionen und Werkzeuge.

Diese klare Trennung gewährleistet, dass Änderungen oder Erweiterungen in einer Komponente minimale Auswirkungen auf andere Komponenten haben.

5.2.2 MVC-Architektur mit JavaFX

Das MVC-Architekturmuster der Anwendung wird durch das JavaFX-Framework unterstützt. Die Views werden durch FXML-Dateien definiert, die eine deklarative Beschreibung der Benutzeroberfläche ermöglichen. So definiert beispielsweise die `MainView.fxml` den Einstiegspunkt der Anwendung. Durch das Attribut `fx:controller` werden die Views mit dem entsprechenden Controller verknüpft. Nach dem gleichen Prinzip lassen sich mit dem Attribut `fx:id` Objektvariablen oder Methoden mit bestimmten grafischen Elementen verbinden.

Die Kommunikation zwischen View und Controller wird durch FXML-Annotationen unterstützt. Mit `@FXML` annotierte Felder und Methoden werden automatisch mit den entsprechenden FXML-Elementen verbunden.

Der Startpunkt der Anwendung ist die `MainApp`-Klasse, die von der JavaFX-`Application`-Klasse erbt. In der `start()`-Methode wird die erste View geladen und der grundlegende Aufbau der Anwendung initialisiert.

Die Navigation zwischen verschiedenen Views wird durch ein zentrales Routing-System gesteuert. Der `NavigationController` fungiert dabei als Router und ermöglicht den Wechsel zwischen den Views. Alle spezifischen Controller erben von einem `AbstractController`, wodurch sie Zugriff auf die Navigationsfunktionalität erhalten, indem eine Methode bereitgestellt wird, die den `NavigationController` für eine beliebige Instanz setzt, die `AbstractController` implementiert.

Der `NavigationController` implementiert zwei Varianten der `switchView()`-Methode:

- Die erste Variante nimmt nur den Pfad der View entgegen, auf die gewechselt werden soll.
- Die zweite Variante ermöglicht zusätzlich die Übergabe von Daten an die Ziel-View.

Der Navigationsprozess läuft dabei wie folgt ab:

1. Die neue View wird über den `FXMLLoader` geladen.
2. Der zugehörige Controller wird extrahiert.
3. Der `NavigationController` wird für den neuen Controller gesetzt, falls dieser von `AbstractController` erbt.
4. Falls Daten übergeben wurden und der Controller das `DataReceiver`-Interface implementiert, werden diese über die `receiveData()`-Methode weitergereicht.
5. Die neue Szene wird in der `Stage` gesetzt.

Die Pfade zu den Views sind zentral in der `ViewRoutes`-Klasse als Konstanten definiert. Dies gewährleistet eine typsichere Navigation und vereinfacht die Wartung der Routenpfade.

Die eigentliche Geschäftslogik ist in Service-Klassen ausgelagert, die von den Controllern verwendet werden. Dies ermöglicht eine klare Trennung zwischen der UI-Logik und der Datenverarbeitung.

Diese Architektur ermöglicht eine modulare Erweiterung der Anwendung. Neue Funktionalitäten können durch das Hinzufügen von FXML-Views, zugehörigen Controllern und entsprechenden Services implementiert werden, ohne bestehende Komponenten zu beeinflussen. Die lose Kopplung zwischen den Komponenten wird durch das Routing-System und die Service-Abstraktion gewährleistet.

5.2.3 Fachliche Bausteine

Um die fachlichen Funktionalitäten der Anwendung klar voneinander abzugrenzen und die Modularität des Systems zu fördern, wurden logische Einheiten definiert. Diese fachlichen Komponenten (im Folgenden vereinfacht als „Komponenten“ bezeichnet) bündeln jeweils eine spezifische Funktionalität, bestehen aus Teilen mehrerer technischer Komponenten und arbeiten unabhängig von anderen Komponenten, wobei definierte Schnittstellen zur Interaktion genutzt werden. Nachfolgend werden die Hauptkomponenten der Anwendung und deren Zusammenspiel erläutert.

Skizzenerstellung

- **Funktion:** Dient der Erstellung und Bearbeitung des Skizzenmodells.
- **Bestandteile:** `DrawView`, `DrawController`, `SketchService` und `SketchModel`.
- **Interaktion:** Nimmt eine Skizze vom Nutzer entgegen und liefert Daten für die Regionorganisation.

Regionorganisation

- **Funktion:** Organisiert die Platzierung von Icons und teilt die Karte in Regionen auf.
- **Bestandteile:** `IconPlacementView`, `IconPlacementController`, `RegionPartitioningService` und `CellModel / CellModelCollection`.
- **Interaktion:** Nutzt die Ergebnisse der Skizzenerstellung und stellt strukturierte Daten für die Kartengenerierung bereit.

Kartengenerierung

- **Funktion:** Generiert die finale Karte und stellt diese dar.
- **Bestandteile:** `ResultView`, `ResultController`, `MapGenerationService` und `GeneratedMapModel`.

- **Interaktion:** Nutzt Daten von der Regionorganisation.

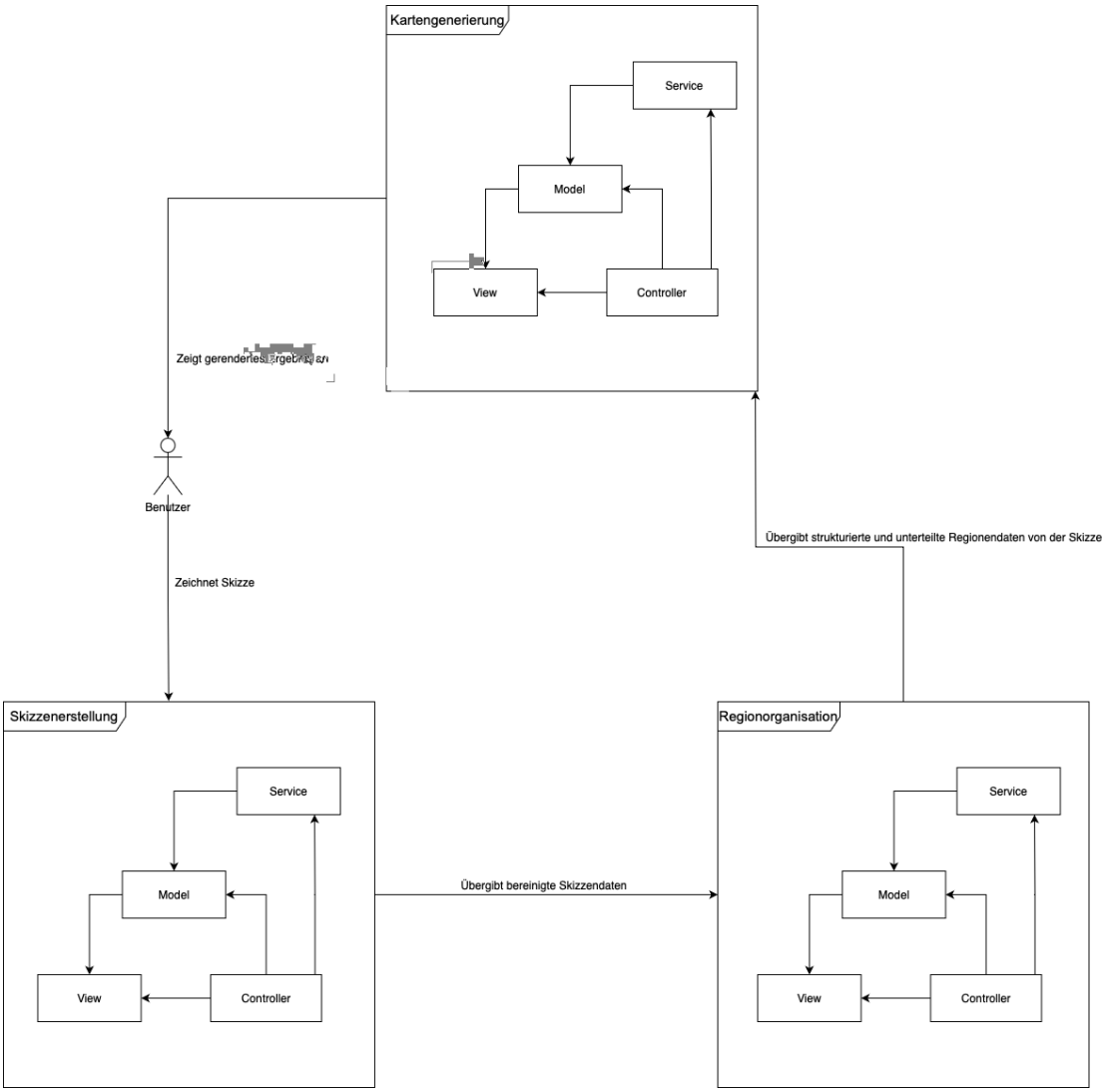

Abbildung 5.1: Bausteinsicht zeigt die drei Hauptkomponenten und stellt das MVC-Muster innerhalb jeder Komponente dar.

5.2.4 Modulare Strategieimplementierung

Das Strategie-Entwurfsmuster gehört zu den Verhaltensmustern und bietet die Möglichkeit, Algorithmen zu kategorisieren, diese innerhalb einer Kategorie während der Laufzeit auszutauschen und sie kontextunabhängig zu nutzen. In der Softwareentwicklung stellt

dieses Muster eine elegante Lösung dar, um verschiedene Implementierungen eines Algorithmus zu kapseln und sie dynamisch austauschbar zu machen.

Für dieses Projekt wird das Entwurfsmuster genutzt, um die Logik für die Generierung einzelner Kartenregionen zu modularisieren. Die einzelnen Regionen, die jeweils mit einem Icon assoziiert werden, besitzen unterschiedliche Eigenschaften. Jede dieser Eigenschaften erfordert eine spezifische Logik zur Generierung der Karteninhalte, die flexibel und erweiterbar sein muss.

Die Strategie-Schnittstelle `MapCellGenerationStrategy` definiert einen einheitlichen Vertrag für alle Generierungsstrategien.

Für jede Region wird eine separate Logik in den folgenden Klassen implementiert:

- `MountainMapCellGenerator`
- `ForestMapCellGenerator`
- `VillageMapCellGenerator`
- `LakeMapCellGenerator`
- `OceanMapCellGenerator`

Die konkreten Implementierungsdetails werden im Abschnitt „Generierung der Kartenelemente“ detailliert beschrieben. Der zentrale Generierungs-Service `MapGenerationService` verwendet eine Zuordnung (`strategyMap`), um die passende Generierungsstrategie für jedes Kartenelement dynamisch auszuwählen. Dies geschieht in der Hauptmethode der Klasse `generateMap()`.

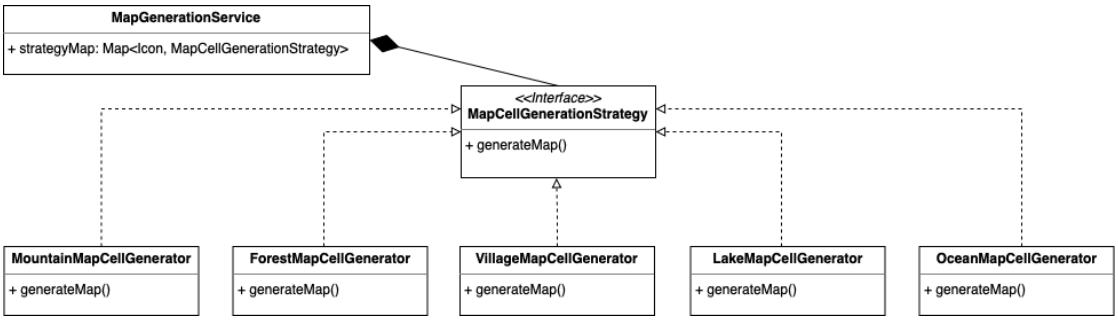

Abbildung 5.2: Klassendiagramm, das die Implementierung des Strategie-Musters in der Anwendung zeigt.

Ablauf der Generierung

Die Methode durchläuft folgende Schritte:

1. **Erstellung des finalen Modells:** Das Modell, das später zur Ausgabe des Ergebnisses genutzt wird, wird instanziiert.
2. **Strategiebasierte Generierung:**
   - Iteration über alle Regionen.
   - Auswahl der passenden Generierungsstrategie.
   - Aufruf der jeweiligen Generierungsmethode.

Vorteile des Ansatzes

Die Vorteile dieses Ansatzes für den Anwendungsfall sind:

- **Erweiterbarkeit:** Neue Landschaftstypen können einfach durch das Hinzufügen einer neuen Strategie-Klasse und eines neuen Icons implementiert werden.
- **Lose Kopplung:** Der Generierungs-Service ist unabhängig von spezifischen Generierungsimplementierungen.
- **Wartbarkeit:** Jede Strategie kann separat entwickelt und getestet werden.
- **Laufzeitflexibilität:** Strategien können dynamisch ausgetauscht oder hinzugefügt werden.

Alternativ wurde ebenfalls ein weiterer Ansatz implementiert, in dem die Generierung selektiv für einzelne Symboltypen erfolgt. Dies geschieht über die überladene `generateMap(Icon icon)`-Methode, die nur für ein bestimmtes Icon die entsprechende Generierungsstrategie anwendet. Ein entscheidender Vorteil dieses Ansatzes besteht darin, dass der Controller die Kartenelemente nun einzeln laden und direkt anzeigen kann. Dadurch entsteht ein interaktiveres Erlebnis für den Nutzer, da sich die Karte schrittweise aufbaut, anstatt erst nach vollständiger Generierung sichtbar zu werden. Der zentrale Nachteil dieses Ansatzes ist die erhöhte Kopplung zwischen Controller und Generierungslogik. Der Controller muss nun aktiv steuern, welche Elemente wann generiert und angezeigt werden, was die Architektur weniger flexibel macht.

5.2.5 Datenfluss

Das MVC-Muster macht es leicht, Daten innerhalb der technischen Komponenten weiterzureichen. Ein Controller kann somit das zugrundeliegende Model einfach anpassen, wodurch dieses wiederum die View verändert. Es wurde bewusst die Entscheidung getroffen, nur triviale Logik in die Model-Klassen aufzunehmen und die Hauptlogik in die Services auszulagern, damit ein Model als Transportmedium für Daten zwischen den fachlichen Komponenten dienen kann, also als Data Transfer Object. Dies verhindert eine unnötig enge Kopplung zwischen Komponenten.

Um Daten zwischen den fachlichen Komponenten zu übertragen, wird ein zusätzliches Konzept eingeführt, das die Kommunikation innerhalb der verschiedenen Systemkomponenten ermöglicht.

Beim Wechsel der Views ruft der zugehörige Controller, sofern er das Interface `DataReceiver` implementiert, automatisch die Methode `receiveData()` auf. Dies ermöglicht eine saubere Trennung der Verantwortung zwischen der View und dem Controller, während gleichzeitig der Controller die Flexibilität behält, auf die übergebenen Daten zu reagieren.

Das Vorgehen lässt sich folgendermaßen beschreiben:

1. **View-Wechsel:** Wenn der Benutzer eine Aktion ausführt, die den Wechsel der aktuellen View auslöst, wird die neue View geladen. Diese View ist mit einem Controller verknüpft, der als `DataReceiver` fungiert.

2. **Datenübergabe:** Sobald die neue View angezeigt wird, werden eventuell vorhandene Daten an den Controller weitergereicht. Dies geschieht durch den Aufruf der Methode `receiveData()`.

3. **Verarbeitung im Controller:** Der Controller, der das `DataReceiver`-Interface implementiert, empfängt die übergebenen Daten und kann sie gemäß den spezifischen Anforderungen der Anwendung verarbeiten. Dies könnte die Aktualisierung von Zuständen, das Füllen von Modellen oder das Auslösen weiterer logischer Prozesse innerhalb der Anwendung umfassen.

Diese Implementierung fördert die Entkopplung zwischen View und Controller, da der Controller lediglich das Interface `DataReceiver` implementiert und nicht direkt von

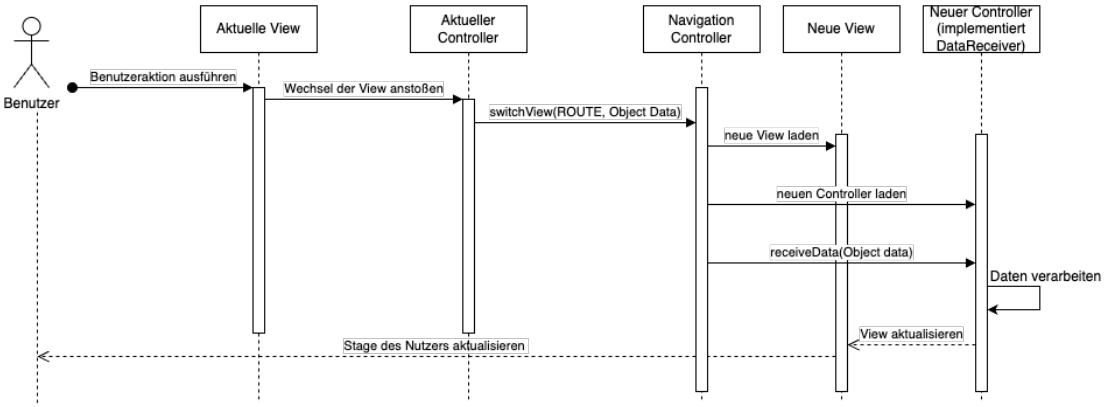

Abbildung 5.3: Sequenzdiagramm, das den beschriebenen Datenfluss veranschaulicht

der View abhängig ist. Die Verantwortung für die Datenverarbeitung liegt klar beim Controller und nicht bei der View.

5.3 Vorverarbeitung

Erstellung der Skizze

Die Erstellung der Skizze bildet den ersten interaktiven Schritt in der Generierungslogik. Entsprechend den Anforderungen wurde eine Lösung entwickelt, die eine intuitive und flexible Zeicheneingabe ermöglicht und gleichzeitig die Komplexität ungenauer oder skizzenhafter Eingaben bewältigt.

Die Sketch-Eingabe wurde als Canvas-Oberfläche implementiert, die dem Nutzer zwei primäre Interaktionsmöglichkeiten bietet: Zeichnen und Radieren.

Der `SketchService` kümmert sich um die Verarbeitung der Skizze, indem er während der Eingabe die gezeichneten Pixel speichert und verwaltet. Dafür interpoliert der Service die Mauspositionen, die er vom Controller erhält, indem er den Bresenham-Algorithmus anwendet. Dieses Vorgehen gewährleistet ein flüssiges Verhalten beim Zeichnen.

Das `SketchModel` repräsentiert die interne Datenstruktur der Skizze. Es speichert gesetzte Pixel als `ClusterableCoordinate` in einer Menge. Die Utility-Klasse `ClusterableCoordinate` erbt von einer Koordinatenrepräsentation aus der Java Topology Suite und implementiert das `Clusterable`-Interface aus Apache Commons.

Wenn der Nutzer mit der Eingabe fertig ist, kann er über die Oberfläche die Verarbeitung der Skizze anstoßen. Dies führt zu folgenden Berechnungen in der Service-Klasse:

1. **Clustering:** Gruppierung zusammenhängender Pixelcluster mit dem DBSCAN-Algorithmus.
2. **Berechnung der Alpha Shape:** Approximation des gezeichneten Umrisses für jedes identifizierte Cluster und gleichzeitige Schließung.

Das erzeugte `SketchModel` wird daraufhin vom aktuellen Controller an den Controller der nächsten View weitergereicht

Segmentierung der Regionen

Ausgehend von der verarbeiteten Skizze wird ein Verfahren benötigt, das die Landkarte in sinnvolle, zusammenhängende Bereiche unterteilt.

Die Segmentierung basiert auf dem `SketchModel` und den Icons, die vom Nutzer beliebig platziert werden können.

Der `RegionPartitioningService` erhält beim Platzieren eines Icons die Koordinaten und das jeweilige Icon vom Controller übergeben und speichert diese.

Nach einer Platzierung wird der Hauptprozess angestoßen:

1. **Voronoi-Diagramm-Generierung:** Auf Grundlage der Icon-Positionen wird zunächst ein Voronoi-Diagramm generiert. Dies erfolgt unter Verwendung einer Delaunay-Triangulation. Anschließend erfolgt das Beschneiden der Zellen anhand des gezeichneten und verarbeiteten Umrisses der Skizze.
2. **CellModel-Erstellung:** Assoziierung einer Zelle mit einem Icon, wobei der gesamte Bereich, der nicht zu einer geschlossenen Fläche der Skizze gehört, dem Ozean-Icon zugeordnet wird.
3. **Hinzufügen zur CellModelCollection:** Die einzelnen `CellModel` werden einer Collection-Klasse hinzugefügt, wobei angrenzende Zellen mit gleichen Icons zusammengefasst werden.

Die entstehende `CellModelCollection`, die eine Sammlung an `CellModel` repräsentiert, wird dem Controller der nächsten View übergeben. Ein `CellModel` besteht aus einem Polygon, einer Farbe für die Visualisierung dieses Zwischenschrittes, einem Icon und der Koordinate des geometrischen Schwerpunkts.

5.4 Generierung der Kartenelemente

Die Implementierungen der `MapCellGenerationStrategy` erhalten über den `MapGenerationService` jeweils ein `CellModel`, das die zu generierende Region repräsentiert, sowie ein `GeneratedMapModel`. Letzteres dient als übergreifendes Model, in dem alle generierten Kartenelemente zusammengeführt werden, um das finale Kartenbild zu erstellen. Das `GeneratedMapModel` besteht hauptsächlich aus einem JavaFX `WritableImage`, das als zentrale Datenstruktur für die kumulierte Kartengenerierung dient. Es speichert sämtliche erzeugten Pixelwerte und Assets, die durch die verschiedenen Strategien der `MapCellGenerationStrategy` hinzugefügt werden, und bringt die generierten Inhalte somit zurück auf die Pixelebene. Dabei funktioniert es ähnlich wie ein Array, das Farbwerte für jede Pixelposition speichert, ist jedoch durch die bereitgestellten Methoden leichter zu handhaben.

5.4.1 Gebirge

Die Generierung von Gebirgsregionen erfolgt unter Verwendung der `MountainMapCellGenerator`-Klasse. Diese Implementierung nutzt ein Perlin Noise Verfahren zur Platzierung von Berg-Assets innerhalb der definierten Zellen. Hierfür durchläuft die Implementierung folgende Schritte:

- **Initialisierung der Parameter:** Faktoren wie die Berggröße, die Rauschschwelle und der minimale Abstand zwischen den Bergen werden definiert.
- **Noise-Generator konfigurieren:** Mit der `FastNoiseLite`-Bibliothek wird ein Perlin Noise Algorithmus genutzt, um natürliche Höhenstrukturen für die Gebirge zu simulieren.
- **Noise-basierte Bergplatzierung:** Basierend auf den Noise-Werten und dem Schwellwert werden Berge nur an geeigneten Stellen platziert, wobei ein Mindestabstand zu bereits gesetzten Bergen eingehalten wird.

- **Visualisierung der Berge:** Je nach berechnetem Noise-Wert wird die Berggröße dynamisch angepasst, um eine realistische Darstellung zu erzielen. Zufällige Spiegelungen und Rotationen erhöhen dabei die optische Variation.

5.4.2 Ozean

Die Generierung der Ozeanregion erfolgt durch die `OceanMapCellGenerator`-Klasse, die einen Simplex Noise Algorithmus nutzt, um organische Tiefenverläufe und sanfte Übergänge zwischen flachen Regionen und tiefem Ozean zu erzeugen. Dabei wird ein abgestufter Farbverlauf verwendet, um die verschiedenen Wassertiefen visuell darzustellen. Die Implementierung folgt diesen Schritten:

- **Initialisierung des Noise-Generators:** Mit der `FastNoiseLite`-Bibliothek wird dieses Mal ein Simplex Noise Algorithmus genutzt.
- **Sanfte Übergänge zu Landmassen:** Ein Distanzfaktor von allen Punkten zur Zellgrenze wird berechnet, um weiche Übergänge zwischen Wasser- und Landbereichen zu schaffen. Dieser Faktor wird mit dem berechneten Noise-Wert kombiniert, um den finalen Höhenwert zu beeinflussen. Dadurch entstehen an den Rändern flachere Küstenregionen, die durch die Kombination von Distanz und Noise-Interpolation erzielt werden.
- **Farbzuweisung anhand der Höhe:** Der finale Höhenwert wird in eine passende Farbe umgewandelt. Durch Interpolation zwischen verschiedenen Blautönen entsteht ein natürlicher Farbverlauf.

5.4.3 Seen

Die Generierung der Seen erfolgt durch die `LakeMapCellGenerator`-Klasse, die eine organische Form für den See innerhalb der gegebenen Zellen erzeugt. Dies geschieht durch die Verwendung einer Kombination aus geometrischen Berechnungen und stochastischen Methoden, um den natürlichen Verlauf eines Sees zu simulieren. Die Implementierung folgt dabei den folgenden Schritten:

- **Maximalen Radius berechnen:** Der maximal erlaubte Radius des Sees wird berechnet, indem der Abstand vom Zentrum der Zelle zu der nächsten Kante des

Polygons ermittelt wird. Dies hilft dabei, die Größe an die Zellform anzupassen und sicherzustellen, dass der See innerhalb der Zelle bleibt.

- **Organische Form erzeugen:** Eine organische Seeform wird generiert, die eine unregelmäßige und natürliche Kontur aufweist. Der See erhält eine variable Form, die durch eine zufällige Variation des Radius an jedem Punkt entlang des Randes des Sees erzeugt wird.
- **Rendering des Sees:** Die Farbe jedes Pixels wird basierend auf seiner Entfernung zum Zentrum des Sees und zum Rand des Sees berechnet. Diese Entfernung wird verwendet, um einen sanften Farbverlauf zwischen tiefem Wasser und Wasseroberfläche zu erzeugen.

5.4.4 Wälder

Die Generierung von Waldregionen erfolgt mithilfe der `ForestMapCellGenerator`-Klasse, die durch den Einsatz des Wave Function Collapse Algorithmus eine realistische und organische Verteilung von Baumtexturen innerhalb der definierten Zellen realisiert. Dabei kommen sowohl deterministische als auch stochastische Elemente zum Einsatz, um eine natürliche Optik zu gewährleisten. Die Implementierung folgt dabei den folgenden Schritten:

- **Initialisierung des Input-Tile-Assets:** Zu Beginn wird ein vorgefertigtes Asset geladen, welches als Grundlage für das spätere Texturmuster dient.
- **Konfiguration des WFC-Models:** Mit Hilfe der `OverlappingModel`-Klasse aus der WFC-Bibliothek von Allison Casey wird aus dem Input-Asset ein Texturmuster generiert. Dabei spielt die Parametrisierung eine entscheidende Rolle, damit es nicht zu Fehlern kommt.
- **Texturierung der Zellregion:** Wurde die Methode des WFC-Models erfolgreich ausgeführt, wird das daraus resultierende Muster in ein `TexturePaint`-Objekt überführt, um dann die gesamte Fläche der Zellregion kachelartig zu texturieren. Die Textur wird hier also aus Performancegründen gegebenenfalls mehrfach verwendet, anstatt direkt ein größeres Ausgabebild mit dem WFC-Algorithmus zu generieren.

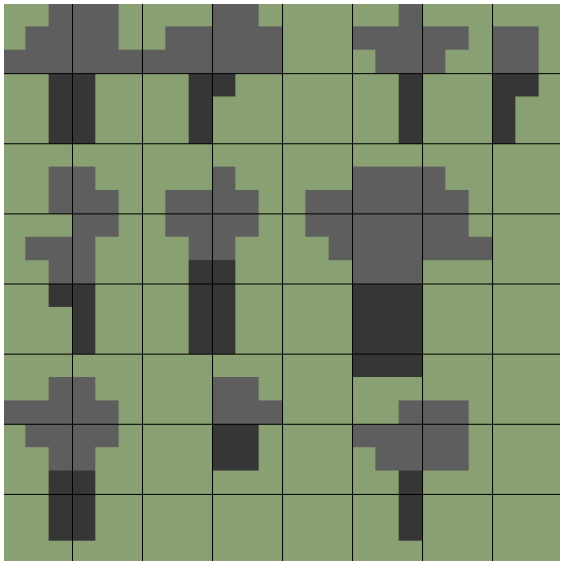

Abbildung 5.4: Eingabebild für den WFC-Algorithmus mit einem 3x3-Raster zur besseren Erkennbarkeit der Struktur

- **Weichzeichnen des Randes:** Bevor die Pixel dem `GeneratedMapModel` hinzugefügt werden, wird zunächst die Entfernung dieses Pixels zur Zellgrenze berechnet. Anhand dieser Distanz wird ein weicher Übergang zwischen der intensiveren Waldtextur im Zellinneren und angrenzenden Regionen erzeugt. Hierbei wird ein Noise-Faktor hinzugefügt, der für eine dynamische Variation und ein natürlicheres Erscheinungsbild sorgt.

5.4.5 Dörfer

Die Generierung der Dorfregionen erfolgt durch die Implementierung der `VillageMapCellGenerator`-Klasse. Hier werden Straßennetze als infrastrukturelles Rückgrat sowie Markierungspunkte mit zufällig generierten Dorfnamen integriert. Der Algorithmus folgt dabei diesen Schritten:

- **Erzeugung eines Straßennetzwerks:** Um die strukturelle Grundlage der Dörfer zu bilden, wird ein Straßennetzwerk in Abhängigkeit von der Größe des Polygons generiert. Hierbei kommen mehrere Teilschritte zum Einsatz:
  - **Seed-Punkt-Generierung:** Eine Anzahl an Seed-Punkten wird relativ zur Fläche des Polygons ermittelt. Diese Seed-Punkte, die zufällig innerhalb des

Polygons platziert werden, fungieren als Knotenpunkte, die später als Anker für das Straßennetz dienen.

  - **Berechnung des minimalen Spannbaums (MST):** Mithilfe von Prim's Algorithmus wird aus den generierten Seed-Punkten ein MST berechnet. Dieser stellt die effizienteste Verbindung zwischen den Punkten dar und bildet somit ein organisch wirkendes Straßennetz [16].

  - **Zeichnen der Straßen:** Die ermittelten Liniensegmente werden pixelweise auf das Model übertragen. Dabei sorgt eine teilweise randomisierte Zeichentechnik dafür, dass die Straßen nicht nur als dünne Linien, sondern mit ausgefransten Kanten und variabler Breite dargestellt werden.

- **Platzierung von Markierungspunkten und Dorfnamen:** Die Knotenpunkte werden als Standorte für Dörfer genutzt und durch leicht versetzte Markierungspunkte visualisiert. Um genug Abstand zwischen den Namen zu gewährleisten, wird geprüft, ob bereits ein Namensschild in der Nähe platziert wurde, bevor ein neuer Name hinzugefügt wird.

- **Zufällige Namensgenerierung:** Zur Beschriftung der Dörfer wird ein Name generiert, der durch die zufällige Kombination von Präfixen, Wortstämmen und Suffixen entsteht.

5.4.6 Details

Neben den bereits beschriebenen Implementierungen der `MapCellGenerationStrategy` umfasst die Kartengenerierung weitere Verarbeitungsschritte, die nicht direkt in den einzelnen Strategien enthalten sind. Der Schwerpunkt dieser zusätzlichen Details liegt in der Nutzung eines stochastischen L-Systems zur prozeduralen Flussgenerierung sowie in der nachgelagerten Bildverarbeitung, welche die finale Ästhetik der Karte beeinflusst.

Nach der initialen Generierung der Zellinhalte wird über den übergeordneten `MapGenerationService` die Landschaft um Flüsse ergänzt. Hierbei wird ein stochastisches L-System verwendet, um organisch wirkende Flussmuster zu erzeugen. Folgende Schritte charakterisieren diesen Prozess:

- **Definition der Regeln:** Für das L-System werden Produktionsregeln mit Gewichtungen definiert, die den Ausbau und die Verzweigung des Flussverlaufs bestimmen.

Für die einzelnen Symbole werden mehrere mögliche Produktionen hinterlegt, aus denen anhand eines Zufallsmechanismus und der jeweiligen Gewichtung jeweils eine Regel ausgewählt wird. Dadurch entsteht bei jedem Durchlauf eine individuelle und variantenreiche Flussgeometrie.

- **Generierung des Strings:** Ausgehend von einem Axiom wird über eine festgelegte Anzahl von Iterationen ein String generiert, der die Anweisungen für den Flussverlauf enthält.
- **Rendering:** Der generierte String wird durch die `RiverTurtleRenderer` Klasse interpretiert. Dieser bewegt sich von einem zufälligen Startpunkt am Rand in Richtung Mitte der Karte, wobei er folgende Besonderheiten aufweist:
  - Die Breite des Flusses wird dynamisch angepasst. Mit zunehmender Entfernung vom Ursprung reduziert sich die Linienbreite, was einen natürlichen Effekt erzeugt und dem Flussverlauf eine realistische Variation verleiht.
  - Zusätzlich werden Kollisionsprüfungen durchgeführt, um zu verhindern, dass der Fluss in als Gebirge markierte Bereiche verläuft.
  - Für jeden Pixel, der im Rahmen des Renderings gesetzt wird, wird die Distanz zur Küstenlinie ermittelt. Diese Distanz dient als Basis für eine interpolierte Farbzuweisung, wodurch ein sanfter Übergang von flachem zu tiefem Wasser entlang der Flussränder erreicht wird. Außerdem wird beim Setzen der Pixel verhindert, dass innerhalb eines Sees gezeichnet wird.

Nachdem alle Landschaftselemente in das `GeneratedMapModel` integriert wurden, erfolgt eine abschließende Nachbearbeitung des Kartenbildes:

- **Rauschfilterung:** Ein gezielt eingesetzter Noise-Filter fügt dem finalen Bild einen subtilen Rauschanteil hinzu. Dies dient dazu, dem Kartenbild eine texturierte Optik zu verleihen und eventuelle Artefakte der prozeduralen Generierung zu kaschieren.
- **Farb- und Textureffekte:** Dem Bild werden Farbnuancen wie ein leichter Gelbstich, Entsättigung und Aufhellung hinzugefügt, um das Bild optisch zu harmonisieren. Zusätzlich wird eine Pergament-Textur mit dem eigentlichen Bild verblendet.
- **Vignettierung:** Ein Vignette-Effekt sorgt an den Rändern für einen dunkleren Übergang. Dieser Effekt verstärkt den Fokus auf die zentralen Kartenelemente.

6 Evaluation

In diesem Kapitel wird die Evaluation des entwickelten Systems dargestellt. Ziel der Evaluation ist es, sowohl die Qualität und Performance des Systems, als auch den angewendeten Entwicklungsprozess kritisch zu überprüfen. Dabei werden die im Konzept definierten funktionalen und nicht-funktionalen Anforderungen als Bezugsrahmen herangezogen.

Diese duale Betrachtungsweise – die Bewertung sowohl des finalen Systems als auch des Entwicklungsprozesses – ermöglicht eine ganzheitliche Beurteilung des Projekts und liefert wichtige Erkenntnisse für zukünftige Arbeiten in diesem Bereich. Insbesondere wird dabei überprüft, ob sich eine strategische Kombination algorithmisch unterschiedlicher Verfahren so gestalten lässt, dass trotz ihrer strukturellen Divergenz eine visuell kohärente und intuitiv steuerbare sketch-basierte Generierung fiktionaler Landkarten ermöglicht wird.

6.1 Methodik

Die Evaluierung des Entwicklungsprozesses basiert auf einem agilen, iterativen und komponentenbasierten Vorgehen. Ziel war es, durch klar abgegrenzte Module und definierte Epics frühzeitig funktionsfähige Systemteile bereitzustellen, die unabhängig voneinander getestet werden konnten. Im Folgenden wird der Vergleich zwischen geplanter Methodik und praktischer Umsetzung dargestellt.

Umsetzung der Entwicklungsphasen und Modularität

Die geplanten Phasen – vom Aufbau eines responsiven Applikationsgerüsts mit Sketch-Eingabe über die Optimierung und icon-basierte Segmentierung bis hin zur prozeduralen Generierung von Landschaftstypen und deren Integration – wurden nahezu vollständig

und ohne wesentliche Abweichungen realisiert. Die von Anfang an festgelegten modularen Grenzen ermöglichten es, Änderungen gezielt auf einzelne Komponenten zu beschränken. Dadurch konnten die fachlichen Komponenten unabhängig voneinander entwickelt und direkt getestet werden.

Architektur und Flexibilität

Die modulare Architektur mit klar definierten Controllern, Services und Models wurde erfolgreich umgesetzt. Dank des gewählten Technologiestacks entstand ein stabiles Grundgerüst, das insbesondere in der Kartengenerierung durch den Einsatz des Strategie-Musters flexible Erweiterungen der Landschaftsalgorithmen ermöglichte. Zwar waren Bereiche wie die Skizzenerstellung und Regionenorganisation weniger flexibel, dennoch konnten spätere Erweiterungen – beispielsweise die Zusammenführung benachbarter Voronoi-Polygone – ohne größere Umstrukturierungen integriert werden, was sich positiv auf die visuelle Darstellung auswirkte.

Kontinuierliches Feedback und Testbarkeit

Um die Qualität der Implementierung zu sichern, wurde ein strukturiertes Testkonzept entwickelt, das auf der modularen Architektur des Systems aufbaut und einen komponentenorientierten Testansatz verfolgt. Die Tests wurden in drei Hauptebenen organisiert:

Komponententests: Jede Komponente wurde isoliert getestet, um ihre korrekte Funktionalität unabhängig von anderen Systemteilen zu validieren:

- **Skizzenerstellung:** Für diese Komponente wurde ein spezieller Visualisierungsmodus implementiert, der die Ergebnisse der Algorithmen zur Umrisserkennung und Formenkorrektur unmittelbar nach dem Zeichenvorgang darstellt. Dies ermöglichte eine direkte visuelle Validierung des Algorithmus und eine zeitnahe Anpassung der Parameter zur Optimierung der Genauigkeit.
- **Regionenorganisation:** Um diese Komponente unabhängig zu testen, wurden vordefinierte Skizzen als Eingabe verwendet, womit der Einstiegspunkt direkt auf die Regionenorganisation gesetzt werden konnte. Ein implementierter Visualisierungsmodus stellte die erzeugten Voronoi-Zellen mit farblicher Differenzierung dar,

was eine effiziente Überprüfung der korrekten Zellenbildung ermöglichte. Diese Visualisierung erwies sich als so wertvoll, dass sie größtenteils in die finale Implementierung übernommen wurde.

- **Kartengenerierung:** Dank des eingesetzten Strategie-Musters konnten die einzelnen Landschaftsalgorithmen isoliert getestet werden, ohne Seiteneffekte auf andere Systemteile befürchten zu müssen.

Integrationstest: Nach der erfolgreichen Validierung der Einzelkomponenten wurden Integrationstests durchgeführt, um das Zusammenspiel der Komponenten zu überprüfen. Hierbei lag der Fokus besonders auf den Schnittstellen:

- **Skizze zu Polygon:** Tests zur Validierung der korrekten Umwandlung von Skizzen in geometrische Strukturen.
- **Regionen zu Landschaft:** Tests zur Sicherstellung der korrekten Anwendung der Landschaftsalgorithmen auf die entsprechenden Regionen mit besonderer Beachtung der Übergangsbereiche.

Systemtests: Abschließend wurden Systemtests durchgeführt, die den gesamten Ablauf von der Skizzeneingabe bis zur fertigen Karte umfassten. Dabei wurden verschiedene Komplexitätsgrade getestet:

- **Einfache Karten:** Test mit wenigen, klar abgegrenzten Regionen.
- **Komplexe Karten:** Test mit vielen, teilweise kleinen Regionen und unterschiedlichen Landschaftstypen.
- **Grenzfälle:** Gezieltes Testen von Extremsituationen, wie sehr detaillierte Umrisse oder besonders hohe Regionendichte.

Der iterative Entwicklungsansatz ermöglichte es, nach jedem Testzyklus unmittelbare Anpassungen vorzunehmen und die Ergebnisse in den nächsten Zyklus einfließen zu lassen. Für zukünftige Weiterentwicklungen wurde ein spezieller Testmodus als sinnvolle Erweiterung identifiziert, der den direkten Zugriff auf relevante Systemteile ermöglichen und so aufwändige Berechnungen für Testzwecke umgehen würde.

Fazit

Der agile, iterative und komponentenbasierte Ansatz hat nicht nur die Umsetzung der geplanten Entwicklungsphasen erleichtert, sondern auch eine flexible und robuste Architektur geschaffen, die Erweiterungen und Anpassungen ermöglicht. Die klare Modularisierung und frühe Testbarkeit der Komponenten haben den gesamten Entwicklungsprozess entscheidend positiv beeinflusst.

6.2 Qualitative und Ästhetische Bewertung

Umriss

Die Abbildung 6.1 zeigt, dass der entwickelte Algorithmus auch detaillierte Formen abbilden kann. Allerdings bleibt die ursprüngliche Form nicht vollständig erhalten. An einigen scharfen Kanten der Zeichnung sind Glättungen erkennbar, die für diesen Anwendungsfall jedoch noch im akzeptablen Rahmen liegen.

Abbildung 6.2 verdeutlicht, dass ungenaue Formen korrigiert werden können. Sind die Lücken jedoch zu groß, führt dies zu Fehlverhalten des Algorithmus, wie in Abbildung 6.3 zu sehen ist. Das negative Beispiel lässt sich auf die Parametrisierung sowie das Zusammenspiel von Formerkennung und Clustering zurückführen.

In Abbildung 6.4 ist zu erkennen, dass der Clustering-Algorithmus zuverlässig arbeitet und auch nah beieinanderliegende Umrisse individuell behandelt. Dennoch gibt es eine zentrale Herausforderung:

- **Schwierige Parametrisierung des DBSCAN-Algorithmus:** Eine der größten Herausforderungen ist die Wahl der Parameter für den DBSCAN-Algorithmus. Ein zu kleiner ϵ-Wert führt dazu, dass zusammengehörige Strukturen fälschlicherweise nicht verbunden werden, während ein zu großer Wert zu einer unerwünschten Zusammenfassung verschiedener Umrisse führen kann.

Eine adaptive oder heuristisch gesteuerte Parametrisierung könnte helfen, die Qualität der Umrisse weiter zu verbessern und die Robustheit des Algorithmus zu erhöhen.

Polygon

Polygone werden mit hoher Genauigkeit generiert und nähern sich dem ursprünglichen Umriss gut an. Allerdings sind in Abbildung 6.5 einige Artefakte sichtbar, die darauf hinweisen, dass das Polygon den Umriss nicht perfekt abbilden kann.

Ozean

Die Darstellung des Ozeans in Abbildung 6.6 zeigt zwar ein realistisches Wechselspiel zwischen tiefem und flachem Wasser, weist jedoch einige Schwächen auf:

- **Geringe Varianz:** Die erzeugte Struktur des Ozeans zeigt nur eine begrenzte Variation in Tiefe und Farbgebung. Dies führt zu einer eher monotonen Darstellung, die nicht die Vielfalt realer Meereslandschaften widerspiegelt. Eine gezielte Anpassung der Parameter könnte hier für mehr visuelle Abwechslung sorgen.
- **Verlust von Details durch Filter:** Die durch Noise Verfahren erzeugten Muster gehen insbesondere nach der Anwendung von Filtern teilweise verloren. Dies kann dazu führen, dass feinere Strukturen, die zur realistischen Darstellung beitragen, nicht mehr deutlich sichtbar sind. Eine Verbesserung könnte durch eine kontrastreichere Farbgestaltung oder eine gezielte Anpassung der Schwellenwerte erfolgen.
- **Fehlende kleinere Details:** Der Ozean wirkt aktuell eher statisch, da kleinere Merkmale wie Wellen oder subtile Oberflächenstrukturen fehlen. Die Integration solcher Details könnte die visuelle Qualität weiter verbessern und die Karte lebendiger erscheinen lassen.

Zusammenfassend bietet die aktuelle Ozean-Darstellung eine solide Grundlage, profitiert jedoch von gezielter Nachbearbeitung und einer feineren Abstimmung der Parameter, um sowohl Varianz als auch Detailgrad zu erhöhen.

Gebirge

Die Gebirge in Abbildung 6.7 erinnern stilistisch an handgezeichnete Karten und weisen durch Verteilung, Größe und Rotation organische Variationen auf. Dank der speziellen Eigenschaften von Perlin Noise wirkt die Verteilung nicht zu zufällig. Allerdings gibt es einige Einschränkungen:

- **Begrenzte Variation durch einzelnes Asset:** Da nur ein einziges Asset für die Gebirgsdarstellung verwendet wurde, fällt die visuelle Vielfalt eingeschränkt aus. Mehrere unterschiedliche Bergformen oder Variationen desselben Grunddesigns könnten hier für eine abwechslungsreichere und natürlichere Darstellung sorgen.
- **Parametrisierungsproblem bei kleinen Zellen:** Aktuell ist die Parametrisierung der Bergplatzierung teilweise linear von der Größe der jeweiligen Zelle abhängig. Dies führt dazu, dass in sehr kleinen Zellen kaum Berge generiert werden. Eine alternative Skalierungsstrategie oder eine nicht-lineare Abhängigkeit könnte helfen, auch bei kleineren Zellen eine angemessene Bergverteilung zu gewährleisten.

Während die aktuelle Gebirgsdarstellung bereits eine ansprechende, handgezeichnete Ästhetik aufweist, könnte eine überarbeitete Parametrisierung sowie eine größere Asset-Vielfalt die visuelle Qualität weiter steigern.

Seen

Seen variieren in Größe und Form. Wie in Abbildung 6.8 zu sehen ist, sind organische Übergänge zur Seemitte und zum Seerand sichtbar. Es gibt jedoch einige Schwächen in der aktuellen Umsetzung:

- **Geringe Formvariation:** Die Seen zeigen eine relativ einheitliche und kreisförmige Struktur, wodurch die visuelle Vielfalt eingeschränkt ist. Eine größere Variation in der Formgebung könnte die Natürlichkeit der Gewässer erhöhen.
- **Unzureichende Anpassung an Zellgeometrie:** Der Algorithmus zur Seegenerierung berücksichtigt derzeit nicht ausreichend die Form der umgebenden Zelle. Bei länglichen Zellen wird der See nicht entsprechend gestreckt, sondern begrenzt seinen Radius auf die Entfernung vom Mittelpunkt zum nächsten Zellrand. Dies kann dazu führen, dass große Zellen dennoch nur kleine Seen enthalten, was inkonsistent wirken kann. Eine Anpassung der Generierungslogik, die stärker die Zellgeometrie einbezieht, könnte hier für eine natürlichere Platzierung sorgen.

Zusammenfassend bietet die aktuelle Seengenerierung zwar eine solide Basis, würde aber von einer verbesserten Formvielfalt und einer stärkeren Anpassung an die Zellstruktur profitieren.

Wälder

Die Wälder weisen eine hohe Variation und eine natürliche Verteilung der Bäume auf, wie in Abbildung 6.9 ersichtlich ist. Dennoch gibt es einige Aspekte, die verbessert werden könnten:

- **Pixelartige Darstellung der Bäume:** Aufgrund der Arbeitsweise des WFC Algorithmus wirken die einzelnen Bäume etwas pixeliger im Vergleich zu anderen Landschaftsdetails. Dies kann zu einem Stilbruch innerhalb der Karte führen. Eine Nachbearbeitung oder Glättung der Baumdarstellung könnte helfen, diesen Effekt abzuschwächen.
- **Wiederholungsmuster bei großen Waldflächen:** Um die Performance zu optimieren, wird der generierte Wald-Output wiederverwendet. Dies kann jedoch dazu führen, dass bei sehr großen Baumzellen sichtbare Wiederholungsmuster entstehen. Eine mögliche Lösung wäre eine adaptive Variation innerhalb des wiederholten Musters, um die erkennbaren Muster zu reduzieren.

Insgesamt bietet die aktuelle Waldgenerierung eine überzeugende visuelle Vielfalt, könnte jedoch durch eine verfeinerte Darstellung der Bäume und eine gezieltere Vermeidung von Wiederholungsmustern weiter verbessert werden.

Dörfer

Abbildung 6.10 zeigt Dörfer sowie deren Verbindungen, die den Eindruck einer pragmatisch gezeichneten Landkarte vermitteln. Dennoch gibt es einige Verbesserungsmöglichkeiten:

- **Probleme bei kleinen oder ungewöhnlich geformten Zellen:** In sehr kleinen oder ungewöhnlich geformten Zellen kann es vorkommen, dass eine Zelle nur aus einem einzelnen Punkt besteht. Dies wirkt unausgewogen und könnte durch eine überarbeitete Parametrisierung verbessert werden, ähnlich wie bei der Gebirgsgenerierung.
- **Optimierung der Punktverteilung:** Die Platzierung der Dörfer erfolgt derzeit zufällig, was zu unregelmäßigen oder unharmonischen Anordnungen führen kann. Eine alternative Methode wie die Verwendung von "Blue-Noise"– ähnlich wie es

Mapgen4 für die Berggenerierung nutzt – könnte hier eine gleichmäßigere und gleichzeitig natürliche Verteilung ermöglichen.

- **Fehlende Details für eine lebendigere Darstellung:** Aktuell bestehen die Dörfer nur aus grundlegenden Strukturen, ohne weitere visuelle Details. Zusätzliche Elemente wie Häuser, Türme oder andere charakteristische Gebäude könnten dazu beitragen, die Karte lebendiger und atmosphärischer wirken zu lassen.

Zusammenfassend bieten die Dörfer eine funktionale und stilistisch passende Darstellung, profitieren jedoch von einer optimierten Parametrisierung, einer verbesserten Punktverteilung und einer höheren Detailtiefe.

Flüsse

Die Flüsse in Abbildung 6.11 wirken zwar zufällig, fügen sich jedoch organisch in die Landschaft ein und zeigen Strukturen, wie man sie aus der Natur kennt. Dennoch gibt es einige Verbesserungsmöglichkeiten:

- **Unnatürliche Anhäufung von Flussquellen:** Da die Verteilung der Flussquellen aktuell zufällig erfolgt, kann es gelegentlich zu unnatürlich wirkenden Anhäufungen kommen. Dies kann dazu führen, dass mehrere Flüsse sehr nah beieinander entspringen. Eine Verbesserung wäre die Nutzung von Heuristiken zur Platzierung der Quellen, beispielsweise basierend auf Höhenkarten oder Niederschlagsmodellen, um eine natürlichere Verteilung zu erreichen.
- **Keine Nutzerkontrolle über die Flussgenerierung:** Der Nutzer hat aktuell keine Möglichkeit, Einfluss auf die Flussgenerierung zu nehmen. Eine anpassbare Parametrisierung, etwa zur Steuerung der Flusslänge, Krümmung oder Verzweigungshäufigkeit, könnte hilfreich sein, um die Generierung besser an individuelle Bedürfnisse oder gewünschte Kartentypen anzupassen.

Durch eine heuristisch gesteuerte Quellplatzierung und eine flexiblere Parametrisierung könnte die Flussgenerierung weiter verbessert werden.

Integration

Abbildung 6.12 illustriert den gesamten Generierungsprozess und das Zusammenspiel aller Landschaftselemente sowie deren Übergänge. Die Evaluation zeigt, dass die Integration der verschiedenen algorithmischen Verfahren trotz ihrer strukturellen Unterschiede erfolgreich umgesetzt wurde. Die Kombination aus geometrischen Algorithmen, unterschiedlichen regelbasierten Verfahren und noise-basierten Methoden erzeugt visuell kohärente Ergebnisse. Die intuitive Steuerbarkeit wird durch die sketch-basierte Eingabe gewährleistet, wie die Beispiele der Umrisserkennung und des Clusterings belegen.

Besonders hervorzuheben ist das Zusammenspiel der unterschiedlichen Verfahren bei den Landschaftsübergängen. Die Flüsse integrieren sich gezielt in ausgewählte Zellen und verlaufen mit weichen Übergängen zu Seen und Küsten. Zwar durchschneiden sie vereinzelt Bäume in Waldregionen, doch fällt dies aufgrund der hohen Baumdichte kaum auf. Insgesamt sind die Übergänge gelungen.

Die Gesamtqualität der generierten Karten zeigt, dass die gewählte Strategie zur Integration der verschiedenen Algorithmen grundsätzlich funktioniert. Dies wird besonders in Abbildung 6.12 deutlich, die das Zusammenwirken aller Komponenten anschaulich darstellt. Die Farbgebung des Gesamtbildes ist stimmig, insbesondere nach Anwendung der Filter, sodass eine harmonische Darstellung der Landschaftselemente erreicht wird.

Dennoch gibt es einige kritische Punkte, die noch verbessert werden können:

- **Abgehackte Flüsse:** Selten, aber manchmal sind die Flüsse in bestimmten Bereichen etwas abgehackt und unterbrechen die fließende Struktur. Dies könnte durch eine bessere Parametrisierung der Quell- und Flussgenerierung verbessert werden.
- **Polygonstrukturen bei kleinen Zellen:** Wenn viele kleine Zellen generiert werden, können Polygonstrukturen mit scharfen Kanten sichtbar werden, was zu einer weniger natürlichen Darstellung führt. Eine "globale Übergangslogik" zwischen den Zellen könnte hier Abhilfe schaffen.
- **Leere Karten bei kleinen Zellen:** Insbesondere bei der Generierung vieler kleiner Zellen, etwa bei einem Wechsel zwischen Bergen und Dörfern, kann die Karte sehr leer wirken, da die Zellen zu klein sind, um ausreichend Inhalte zu generieren. Dies könnte durch eine feinere Anpassung der Zellenparameter oder durch das Hinzufügen zusätzlicher Landschaftsstrukturen behoben werden.

- **Optimierung der Übergänge von Waldregionen:** Es besteht Optimierungspotenzial, insbesondere zwischen Waldregionen und anderen Landschaftselementen, da hier lediglich ein Weichzeichnungs-Effekt verwendet wurde.
- **Abgeschnittene Assets:** Stellenweise erscheinen Berge an Küsten abgeschnitten. Dies könnte durch eine genauere Prüfung der Polygonränder und eine verbesserte Anpassung an angrenzende Landschaftselemente optimiert werden.

Durch die Verbesserung dieser Punkte könnte die Gesamtqualität der generierten Karten weiter optimiert und das visuelle Erlebnis noch realistischer gestaltet werden.

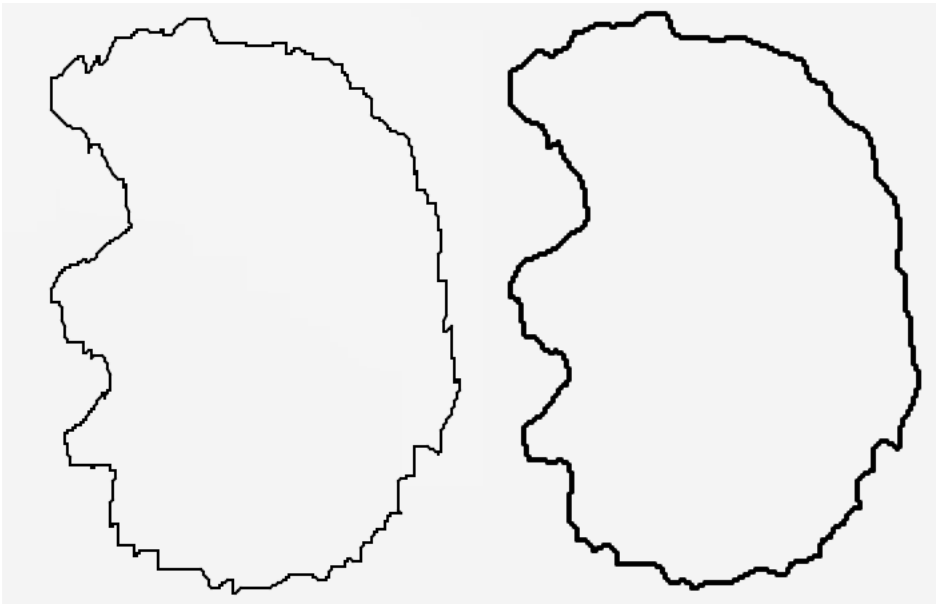

Abbildung 6.1: Links ist die ursprüngliche Eingabe, rechts der verarbeitete Umriss.

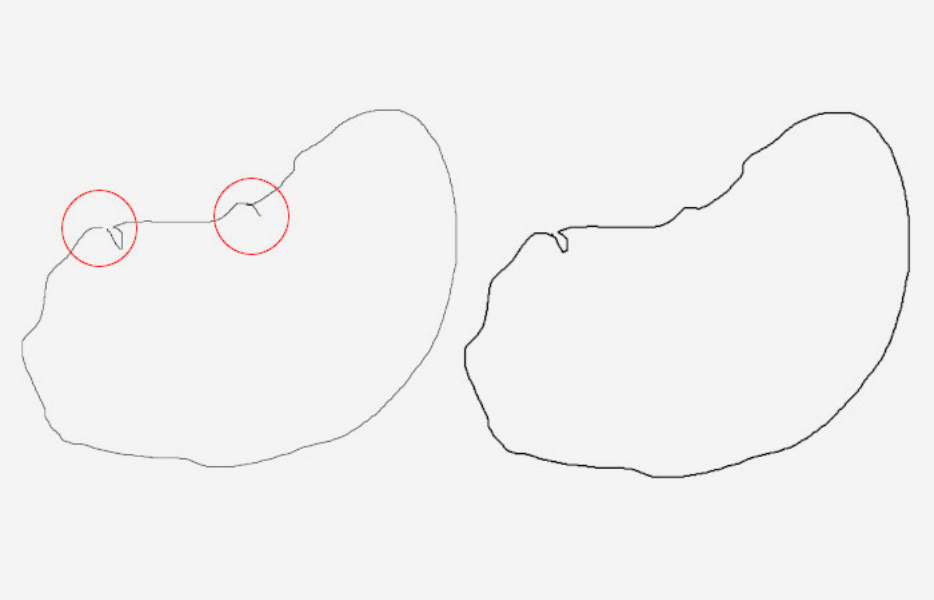

Abbildung 6.2: Links ist die ungenaue Eingabe, rechts der korrigierte Umriss

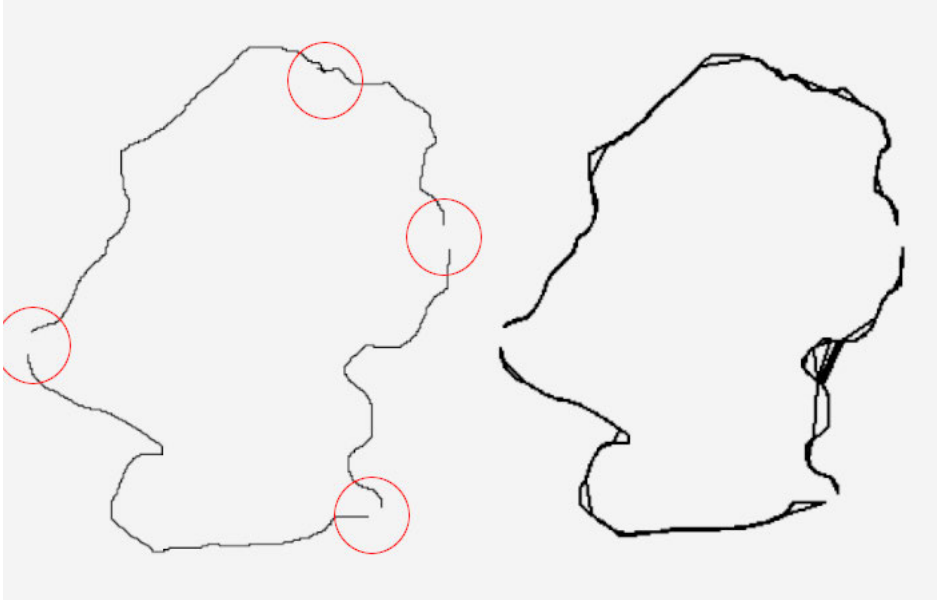

Abbildung 6.3: Links ist die ungenaue Eingabe, rechts der fehlerhafte Umriss.

Abbildung 6.4: Links ist die Eingabe, rechts die individuellen Umrisse.

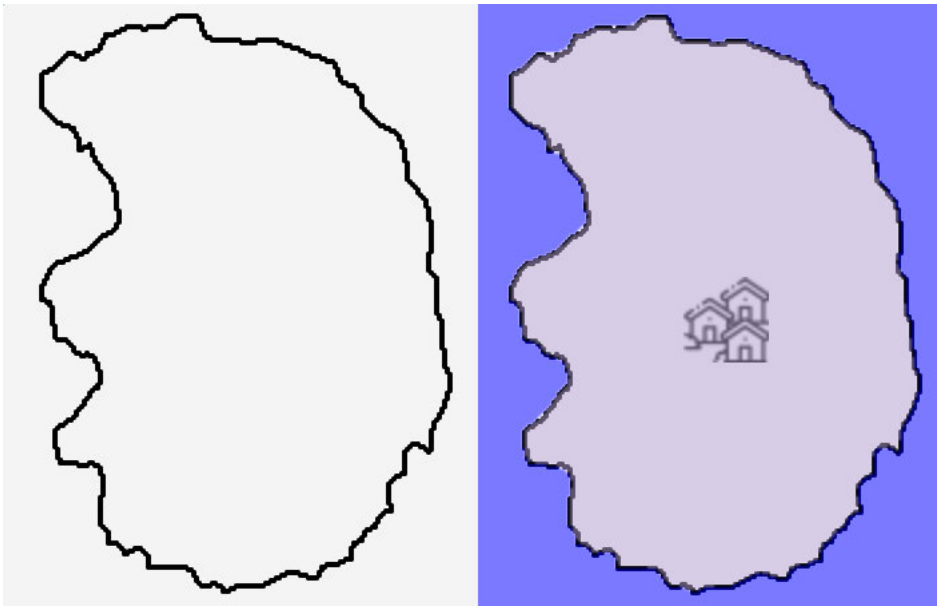

Abbildung 6.5: Links ist der verarbeitete Umriss, rechts das Polygon.

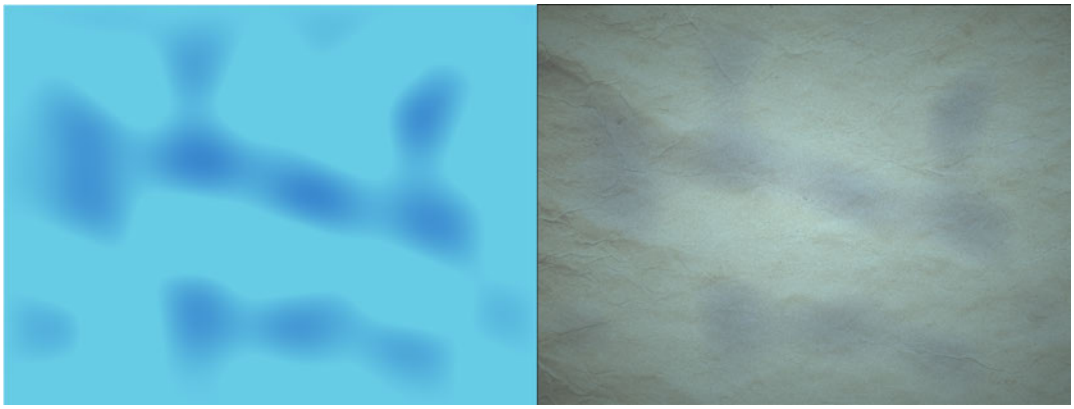

Abbildung 6.6: Links eine generierte Ozeanregion ohne Filter, rechts mit Filter

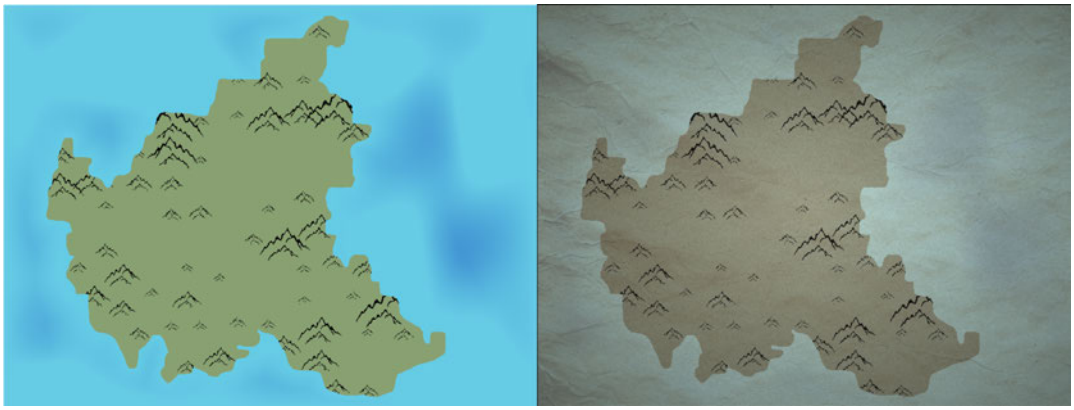

Abbildung 6.7: Links ein generiertes Gebirge ohne Filter, rechts mit Filter

Abbildung 6.8: Links ein generierter See ohne Filter, rechts mit Filter

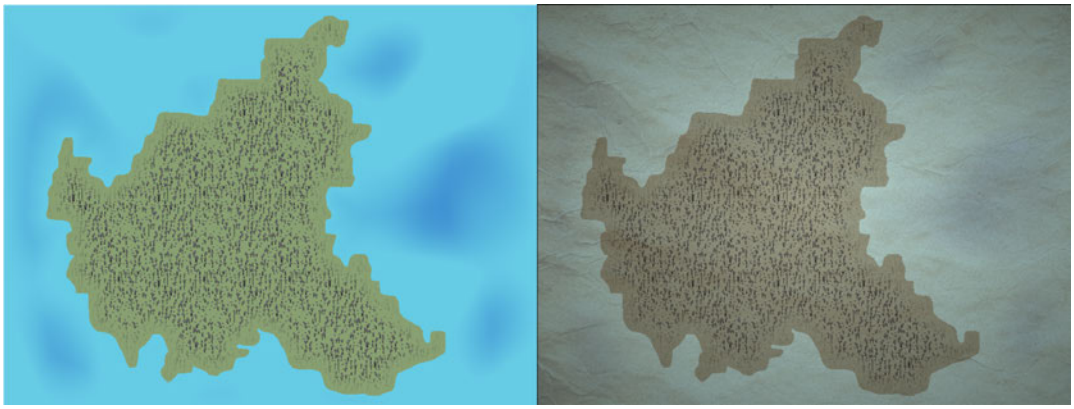

Abbildung 6.9: Links ein generierter Wald ohne Filter, rechts mit Filter

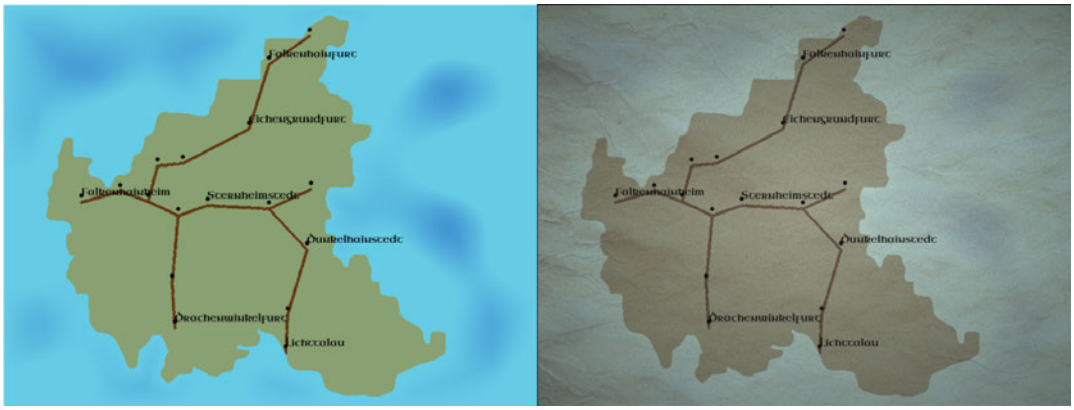

Abbildung 6.10: Links ein generiertes Dorf ohne Filter, rechts mit Filter

Abbildung 6.11: Links generierte Flüsse ohne Filter, rechts mit Filter

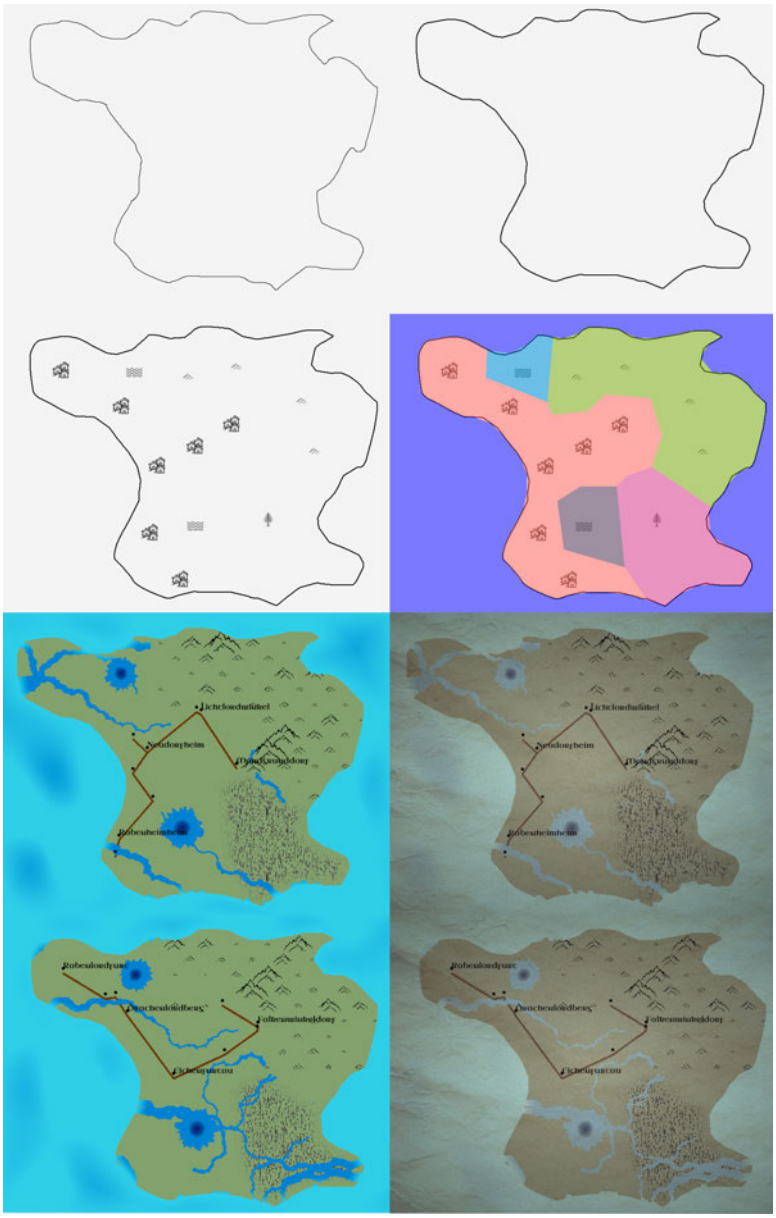

Abbildung 6.12: Zeigt den gesamten Prozess vom Zeichnen über die Verarbeitung bis zur generierten Karte, wobei auf Grundlage derselben Polygone zweimal generiert wurde, um die Unterschiede der Ergebnisse hervorzuheben.

6.3 Performance-Analyse

Das System konnte während der Entwicklung an einigen Stellen hinsichtlich der Performance optimiert werden, vor allem durch die Ersetzung anfänglicher primitiver Datenstrukturen durch besser geeignete. Ein besonders effektiver Optimierungsschritt war die Implementierung von `PreparedGeometries` für die Polygon-Berechnungen. Dennoch lässt sich vorwegnehmen, dass das System und die zugrundeliegenden Berechnungen nicht die Laufzeiten erreichen, die für eine Echtzeitanwendung erforderlich gewesen wären – was jedoch auch kein gesetztes Ziel war.

Im Folgenden sind die durchschnittlichen Berechnungszeiten für einzelne Schritte und Algorithmen angegeben, wobei die Zeitangaben für Karten mit einer Größe von 800x600 Pixeln gelten und das System auf einem M1 MacBook Air ausgeführt wurde. Die unterschiedlichen Zellen haben für die Tests eine typische und gleiche Größe angenommen. Die Tabelle (6.1) zeigt dabei Messungen einer naiven Implementierung im Vergleich zu einer optimierten Version, bei der `PreparedGeometries` zum Einsatz kamen.

Schritt/Algorithmus	**Naiv**	**Optimiert**
Umriss aus Skizze bilden	190	190
Polygone berechnen (bei 20 platzierten Icons)	180	10
Voronoi-Polygone zeichnen (bei 20 platzierten Icons)	7610	210
Dorf berechnen und zeichnen	112	3
Gebirge berechnen und zeichnen	135	58
Wald berechnen und zeichnen	1110	918
See berechnen und zeichnen	292	35
Ozean berechnen und zeichnen	62	62
Flüsse berechnen und zeichnen	24	24

Tabelle 6.1: Vergleich der Messungen zwischen naiver und optimierter Implementierung (`PreparedGeometries`)

Die Optimierung durch `PreparedGeometries` zeigt beeindruckende Verbesserungen in mehreren Bereichen. Besonders deutlich wird dies beim Zeichnen der Voronoi-Polygone, wo die Ausführungszeit von 7610 ms auf 210 ms reduziert werden konnte - eine Verbesserung um den Faktor 36.

Ein wesentlicher Grund für die ursprünglich schlechte Performance lag in der häufigen Verwendung der `contains()`-Methode direkt auf die Polygon Objekte. Diese Operation ist rechenintensiv, da sie für jeden zu prüfenden Punkt komplexe geometrische Berechnungen durchführen muss. Durch die Verwendung von `PreparedGeometries` konnte

diese Operation erheblich beschleunigt werden, da diese eine optimierte interne Datenstruktur bereitstellen.

Dank der Performance-Verbesserungen ist es nun möglich, das Canvas nach jeder Platzierung eines Icons zu aktualisieren. Dadurch erhält der Nutzer sofortiges und schnelles visuelles Feedback, was die Anwendung deutlich interaktiver macht.

Trotz dieser signifikanten Optimierungen bleiben einige Performance-Engpässe bestehen. Das Zeichnen der Voronoi-Polygone benötigt immer noch mehr Zeit als die eigentliche Berechnung, da über alle Zellen iteriert werden muss. Innerhalb dieser Schleife wird erneut über die Pixelkoordinaten innerhalb der Bounding Box der jeweiligen Zelle iteriert. Die Waldgenerierung bleibt mit 918 ms weiterhin der zeitaufwändigste Algorithmus unter den Landschaftselementen.

Zusammenfassend lässt sich sagen, dass durch den Einsatz von `PreparedGeometries` erhebliche Performance-Verbesserungen erzielt werden konnten. Für moderate Kartengrößen sind die Wartezeiten nun deutlich akzeptabler, wenn auch weiterhin keine Echtzeitanwendung erreicht werden kann. Die verbleibenden Performance-Einschränkungen sind hauptsächlich auf die grundlegende Komplexität der verwendeten Algorithmen zurückzuführen.

6.4 Erfüllungsgrad der Anforderungen

In diesem Abschnitt wird der Erfüllungsgrad der definierten funktionalen und nichtfunktionalen Anforderungen bewertet. Tabelle 6.2 zeigt der Vollständigkeit halber nochmals alle konkreten Anforderungen auf und gibt an, ob diese erfüllt wurden. Wie bereits gezeigt, wurden sämtliche Anforderungen erfolgreich umgesetzt, wobei einige Punkte nicht optimal umgesetzt wurden, ohne jedoch die Erfüllung der Mindestanforderungen zu beeinträchtigen. Diese Punkte bieten Potenzial für zukünftige Verbesserungen.

#	Anforderung	Erfüllt	Nicht erfüllt
1	Pixelgenaues Einlesen von Zeichnungen	✓	
2	Zeichnungen sind korrigierbar	✓	
3	Automatische Verbindung von Linien mit kleinen Unterbrechungen	✓	
4	Details bleiben nach Verarbeitung erhalten (max. 5px Abweichung)	✓	
5	Individuelle Behandlung von Inselstrukturen	✓	
6	Validierung der Icons	✓	
7	Icons unterteilen die Fläche in Regionen	✓	
8	Generierung stilisierter Landschaften basierend auf Icons	✓	
9	Realistische Übergänge zwischen Landschaften	✓	
10	Bekannte Kartendetails (Flüsse, Pergament-Optik, Dorfnamen)	✓	
11	Einzigartige Ergebnisse auch bei identischen Eingaben	✓	
12	Gesamte Generierung als einzelnes Bild gerendert	✓	
13	Export als PNG oder JPG	✓	
14	Echtzeitanzeige der Skizze	✓	
15	Segmentierung in max. 1 Sekunde	✓	
16	Landschaftsgenerierung in max. 5 Sekunden	✓	
17	Keine Überlastung bei Karten mit bis zu 20 Icons	✓	
18	Modularer Aufbau für einfache Erweiterungen	✓	
19	Jedes Modul erfüllt eine klar abgegrenzte Aufgabe	✓	
20	Geringe Kopplung für bessere Wartbarkeit	✓	
21	Flexible Anpassung durch Abstraktionen und Schnittstellen	✓	
22	Steuerung über Parameter ohne Codeänderung	✓	
23	Gut dokumentierter Code für langfristige Verständlichkeit	✓	

Tabelle 6.2: Übersicht der Anforderungen und deren Erfüllung

6.5 Zukunftsperspektiven und Erweiterungsmöglichkeiten

Während der qualitativen und ästhetischen Bewertung des Systems wurden bereits bestehende Verbesserungspotenziale des Systems aufgezeigt. In diesem Abschnitt werden nun weiterführende Erweiterungen behandelt, die mit der zukünftigen Weiterentwicklung des Systems verbunden sind.

Das System bietet in verschiedenen Bereichen Potenzial für Erweiterungen und Anpassungen. Dank seiner Modularität kann an vielen Stellen neuer Code hinzugefügt werden, ohne bestehende Implementierungen zu verändern. Es lassen sich beliebig viele Landschaftstypen integrieren oder die Algorithmen bestehender Typen anpassen. Beispielsweise könnte eine alternative Implementierung für Dörfer eingeführt werden, welche wie die Wald Generierung den WFC-Algorithmus nutzt. In diesem Fall müsste lediglich die zentrale Generierungsklasse angepasst werden.

Im Kapitel “Stand der Technik” wurden verschiedene bestehende Systeme untersucht, die unterschiedliche Herangehensweisen zur Landschaftsgenerierung bieten. Im Vergleich dazu weist das hier vorgestellte System einige Gemeinsamkeiten, aber auch Herausforderungen auf, die durch Erweiterungen adressiert werden könnten.

Diese Systeme nutzen Noise-Algorithmen, um Höhenkarten zu generieren. Diese Höhenkarten können als Grundlage für eine 3D-Visualisierung der Landschaft dienen, indem die generierten Höhenwerte zur Modellierung eines dreidimensionalen Terrains verwendet werden. Eine ähnliche Erweiterung könnte in das hier vorgestellte System integriert werden, um eine interaktive 3D-Ansicht zu ermöglichen.

Sketch2Map nutzt Generative Adversarial Networks, um aus Skizzen topografische Karten zu generieren. Dies könnte als Inspiration dienen, um das aktuelle System um maschinelles Lernen zu erweitern. Beispielsweise könnte ein neuronales Netz trainiert werden, um skizzierte Elemente wie Berge oder Flüsse automatisch zu erkennen und in die Generierung einfließen zu lassen. Dies würde eine noch intuitivere Bedienung ermöglichen und die Icon-Platzierung ersetzen.

Ein weiterer Ansatz zur Verbesserung des Systems wäre, die Performance näher an die Anforderungen einer Echtzeitanwendung heranzuführen. Diese Anpassungen sind zwar komplexer als Landschaftserweiterungen, können jedoch gezielt für einzelne Komponenten vorgenommen werden.

Aktuell werden die Polygone der Zellen größtenteils einzeln verarbeitet. Hier könnte ein Multithreading-Ansatz verwendet werden, um die Landschaftsgenerierung zu parallelisieren. Dazu müsste jedoch die Datenstruktur überarbeitet werden, um Konflikte beim gleichzeitigen Lesen und Schreiben durch mehrere Threads zu vermeiden.

Die individuelle Betrachtung der Zellpolygone hat den Nachteil, dass es während der Generierungsphase schwierig ist, globale Informationen, wie Nachbarschaftsbeziehungen, in den Prozess einzubeziehen. Um dies zu ermöglichen, müsste entweder die zugrundeliegende Datenstruktur geändert oder die Architektur, die die Zellen strikt voneinander trennt, angepasst werden.

7 Fazit

Die vorliegende Arbeit präsentiert ein prototypisches System zur sketch-basierten Generierung fiktionaler Landkarten, das geometrische Algorithmen und prozedurale Methoden kombiniert, um intuitiv gestaltbare und visuell überzeugende Ergebnisse zu erzielen.

Die zentrale Frage, *ob eine strategische Kombination unterschiedlicher algorithmischer Verfahren eine visuell kohärente und intuitiv steuerbare, sketch-basierte Generierung fiktionaler Landkarten ermöglicht*, konnte beantwortet werden. Es wurde gezeigt, dass diese Kombination prinzipiell möglich ist und zu überzeugenden Ergebnissen führt.

Die Arbeit demonstriert zudem, wie die Kombination aus Benutzerinteraktion und algorithmischer Verarbeitung einen effizienten Mittelweg zwischen kreativer Gestaltung und technischer Automatisierung bietet. Dies ermöglicht die Erstellung von fiktiven Karten, die sowohl stilisierte Ästhetik als auch geografische Plausibilität vereinen. Das System besitzt die Fähigkeit natürliche Übergänge zwischen Biomen zu erzeugen und durch stochastische Elemente einzigartige Ergebnisse auch bei identischen Eingaben zu gewährleisten.

Gleichzeitig wurden jedoch auch Schwächen des Systems identifiziert, die in zukünftigen Arbeiten adressiert werden können. Insbesondere zeigte sich Optimierungsbedarf bei der Handhabung ungenauer Eingaben sowie in der Echtzeitperformance des Systems, vor allem bei großen Bildern. Zudem könnten fortgeschrittene Verfahren zur Formerkennung und zur semantischen Interpretation von Skizzen die Benutzerfreundlichkeit weiter verbessern. Ein weiterer potenzieller Verbesserungsbereich liegt in der Verwendung globaler Übergangstechniken, die zu einer harmonischeren Integration der verschiedenen Landschaftstypen führen könnten. Ihre Implementierung ist jedoch aufgrund der derzeit stark modularisierten Architektur nicht trivial und würde eine umfassendere Anpassung des Systemaufbaus erfordern.

Diese Limitationen bieten spannende Ansatzpunkte für weiterführende Forschungsprojekte. Durch den Einsatz zusätzlicher Optimierungsstrategien, den Ausbau der algorith-

mischen Pipeline sowie die Integration neuer Interaktionsmethoden könnte die Effizienz und Präzision der Kartengenerierung noch weiter gesteigert werden. Auch die Erweiterung des Systems um neue prozedurale Techniken oder maschinelle Lernverfahren könnte die Qualität der generierten Karten weiter erhöhen.

Insgesamt eröffnet die vorliegende Arbeit nicht nur neue Perspektiven für die prozedurale Generierung fiktionaler Landkarten, sondern legt auch die Basis für zukünftige Verbesserungen, die das System robuster, flexibler und benutzerfreundlicher machen könnten.

Literaturverzeichnis

[1] Bowyer, A.: Computing Dirichlet tessellations. In: *The Computer Journal* 24 (1981), Nr. 2, S. 162–166

[2] Bresenham, J. E.: Algorithm for computer control of a digital plotter. In: *IBM Systems Journal* 4 (1965), Nr. 1, S. 25–30

[3] Delaunay, B.: Sur la sphère vide. À la mémoire de Georges Voronoï. In: *Izvestiya Akademii Nauk SSSR, Seriya Matematicheskaya* (1934), S. 793–800

[4] Dustler, M. ; Bakic, P. ; Petersson, H. ; Timberg, P. ; Tingberg, A. ; Zackrisson, S.: Application of the fractal Perlin noise algorithm for the generation of simulated breast tissue. In: *Progress in Biomedical Optics and Imaging - Proceedings of SPIE* 9412 (2015)

[5] Edelsbrunner, H. ; Kirkpatrick, D. ; Seidel, R.: On the shape of a set of points in the plane. In: *IEEE Transactions on Information Theory* 29 (1983), Nr. 4, S. 551–559

[6] Elshakhs, Y. S. ; Deliparaschos, K. M. ; Charalambous, T. ; Oliva, G. ; Zolotas, A.: A Comprehensive Survey on Delaunay Triangulation: Applications, Algorithms, and Implementations Over CPUs, GPUs, and FPGAs. In: *IEEE Access* 12 (2024), S. 12562–12585

[7] Ester, M. ; Kriegel, H.-P. ; Sander, J. ; Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, AAAI Press, 1996 (KDD'96), S. 226–231

[8] Gain, J. ; Marais, P. ; Strasser, W.: Terrain sketching. In: *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, Association for Computing Machinery, 2009 (I3D '09), S. 31–38. – ISBN 9781605584294

[9] Gamma, E. ; Helm, R. ; Johnson, R. ; Vlissides, J.: *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., 1995. – ISBN 0201633612

[10] Gustavson, S.: Simplex noise demystified. (2005)

[11] Karth, I. ; Smith, A. M.: WaveFunctionCollapse is constraint solving in the wild. In: *Proceedings of the 12th International Conference on the Foundations of Digital Games*, Association for Computing Machinery, 2017 (FDG '17). – ISBN 9781450353199

[12] Lee, D. ; Schachter, B.: Two Algorithms for Constructing a Delaunay Triangulation. In: *International Journal of Parallel Programming* 9 (1980), 06, S. 219–242

[13] Liebling, T. ; Pournin, L.: Voronoi diagrams and Delaunay triangulations: ubiquitous siamese twins. In: *Documenta Mathematica* 16 (2012)

[14] Perlin, K.: An image synthesizer. In: *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, Association for Computing Machinery, 1985 (SIGGRAPH '85), S. 287—-296. – ISBN 0897911660

[15] Perlin, K.: Improving noise. In: *ACM Trans. Graph.* 21 (2002), Nr. 3, S. 681–682

[16] Prim, R. C.: Shortest connection networks and some generalizations. In: *The Bell System Technical Journal* 36 (1957), Nr. 6, S. 1389–1401

[17] Prusinkiewicz, P. ; Lindenmayer, A.: *Graphical modeling using L-systems*. S. 1–50. In: *The Algorithmic Beauty of Plants*, Springer New York, 1990. – ISBN 978-1-4613-8476-2

[18] Shaker, N. ; Togelius, J. ; Nelson, M. J.: *Procedural Content Generation in Games*. 1. Springer Cham, 2016 (Computational Synthesis and Creative Systems). – 237 S. – ISBN 978-3-319-42714-0

[19] Talgorn, F.-X. ; Belhadj, F.: Real-Time Sketch-Based Terrain Generation. In: *Proceedings of Computer Graphics International 2018*, Association for Computing Machinery, 2018 (CGI 2018), S. 13—-18. – ISBN 9781450364010

[20] Ulichney, R. A.: Dithering with blue noise. In: *Proceedings of the IEEE* 76 (1988), Nr. 1, S. 56–79

[21] Valencia-Rosado, L. ; Guzman-Zavaleta, Z. ; Starostenko, O.: A Modular Generative Approach for Realistic River Deltas: When L-Systems and cGANs Meet. In: *IEEE Access* PP (2022), S. 1

[22] Voronoi, G.: Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Premier mémoire. Sur quelques propriétés des formes quadratiques positives parfaites. In: *Journal für die reine und angewandte Mathematik (Crelles Journal)* 1908 (1908), Nr. 133, S. 97–102

[23] Wang, T. ; Kurabayashi, S.: Sketch2Map: A Game Map Design Support System Allowing Quick Hand Sketch Prototyping. In: *2020 IEEE Conference on Games (CoG)*, 2020, S. 596–599

A Anhang

A.1 Verwendete Hilfsmittel

In der Tabelle A.1 sind die im Rahmen der Bearbeitung des Themas der Bachelorarbeit verwendeten Werkzeuge und Hilfsmittel aufgelistet.

Tabelle A.1: Verwendete Hilfsmittel und Werkzeuge

Tool	Verwendung
LaTeX	Textsatz- und Layout-Werkzeug verwendet zur Erstellung dieses Dokuments
GIMP	Bearbeitung und Erstellung von Bildern, insbesondere Assets und Abbildungen
DALL·E	Entwurf von Assets, wie beispielsweise der Parchment-Textur
GitHub Copilot	Code Completion für weniger komplexe Hilfsmethoden
ChatGPT	Für Rechtschreib- und Grammatikprüfung genutzt (Prompt: „Überprüfe Rechtschreibung und Grammatik: . . . “)
HAW GitLab	Versionsverwaltung

Erklärung zur selbständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original