

Diplomarbeit

Softwaretechnik in der Spieleentwicklung

Architektur zur automatisierten Spielweltgenerierung

Damian Schmidt

April 2012

Freie Universität Berlin
Institut für Informatik
AG Softwaretechnik

Betreuer:

Prof. Dr. Lutz Prechelt

Prof. Dr. Marco Block-Berlitz

Eidesstattliche Erklärung

Ich versichere an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt habe. Alle wörtlich oder sinngemäß den Werken anderer Autoren entnommenen Stellen sind unter genauer Angabe der Quelle als solche kenntlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Berlin, den 06.04.2012

Damian Schmidt

Inhaltsverzeichnis

1 Motivation und Einführung	1
1.1 Motivation	1
1.2 Einführung	3
2 Theorie und verwandte Arbeiten	4
2.1 Softwaretechnik	4
2.1.1 Definition und Abgrenzung	4
2.1.2 Bedeutung der Softwaretechnik als Wissenschaftsgebiet	5
2.2 Spieleanthropologie	5
2.2.1 Terminologie	5
2.2.2 Definition und Abgrenzung	9
2.2.3 Andere Definitionen	10
2.3 Softwaretechnik und Spieleanthropologie	11
2.3.1 Spiel und Software	12
2.3.2 Nutzenmaßstab der Spieleanthropologie	15
2.3.3 Einordnung von Softwaretechnik und Spieleanthropologie	15
2.4 Fazit	18
3 Projekt Spielweltgenerator	19
3.1 Ziel und Methodik	19
3.1.1 Zielsetzung	19
3.1.2 Nutzen	19
3.1.3 Abgrenzung und Kompromisse	20
3.1.4 Stand von Industrie und Wissenschaft	21
3.1.5 Methodik	21
3.1.6 Lösungsansatz	21
3.2 Modell und Entwurf	23
3.2.1 Analyse	24
3.2.2 Räumliches Metamodell	25
3.2.3 Architektur und Komponenten	28
3.3 Implementierung	37
3.3.1 Metadaten	37
3.3.2 Interpreter	39
3.3.3 Renderer	40
3.3.4 Spielwelt-Daten	41
3.3.5 Bedienschnittstelle	43

3.3.6	Parametrisierung und Zufall	47
3.3.7	Beispielkonfiguration	48
3.3.8	Performance	50
4	Auswertung	56
4.1	Einsatz von Softwaretechnik	56
4.2	Datentrennung in der Architektur	57
4.3	Erweiterungsmöglichkeiten	58
4.3.1	Erweiterungen der Implementierung	58
4.3.2	Erweiterungen des Modells	59
4.4	Ergebnis und Fazit	62
4.4.1	Ergebnis	62
4.4.2	Fazit	64
5	Zusammenfassung und Ausblick	65
5.1	Zusammenfassung	65
5.2	Ausblick	66
Anhang		67
Glossar		71
Literaturverzeichnis		74

1 Motivation und Einführung

Softwaretechnik und Spieleentwicklung bezeichnen Themengebiete, die Schnittmengen aufweisen. Ziel dieser Arbeit ist es, in der Theorie eine genaue Einordnung der beiden Gebiete zueinander vorzunehmen und durch das Projekt anhand von Entwurf und Implementierung einer Architektur zur automatisierten Spielweltgenerierung die Erkenntnisse über die Bedeutung von Entwurfsmustern in der Spieleentwicklung zu erweitern.

1.1 Motivation

Die Spieleentwicklung hat sich seit den 90er Jahren mit dem rasanten Wachstum der Spieleindustrie zunehmend professionalisiert und der Produktionsaufwand einiger Spiele ist längst mit dem von aufwändig produzierten Kinofilmen vergleichbar. Der weltweite Umsatz der Spieleindustrie weist seit Jahren hohe Wachstumsraten auf und hat bereits den der Musikindustrie übertroffen ([32], S. 10).

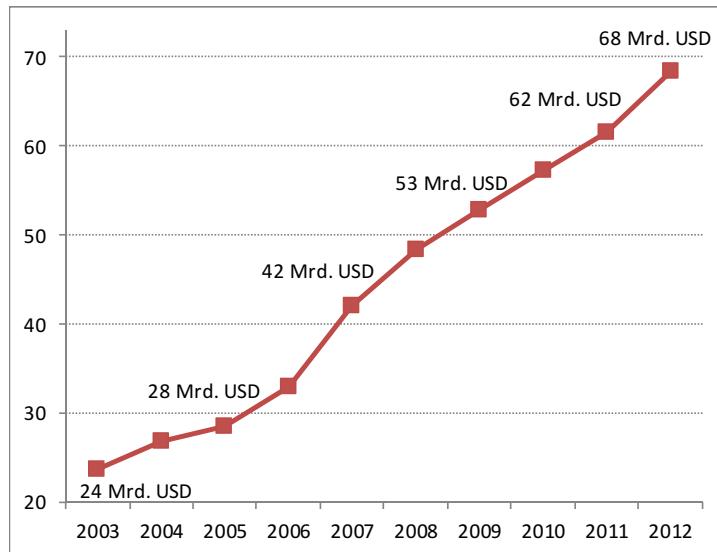


Abbildung 1.1: Weltweiter Videospiele-Umsatz in Mrd. USD, Quelle: PwC (2008)[32]

Spieleindustrie und Spieleentwicklung sind einer besonderen Dynamik aus verschiedenen Einflüssen von Wirtschaft und Technik wie z. B. sich schnell verschiebender

Märkte oder neuartiger Technologien ausgesetzt, spielen in bestimmten Bereichen jedoch auch eine Art Vorreiterrolle und üben so selbst Einfluss auf die Entwicklung anderer Gebiete aus.

Neu ist die Spieleentwicklung als Wissenschaftsgebiet, sodass erst seit den 2000er Jahren verbreitet Ausbildungsberufe und Studiengänge zu den in der Spieleentwicklung üblichen Arbeitsfeldern angeboten werden. Folglich verwundert es nicht, dass die Arbeitswelt der Spieleentwicklung momentan immer noch eine hohe Quereinstiegerquote aufweist (vgl. [17], S. 710; [26], S. 84). Außer Themengebieten, mit denen sich auch die Softwaretechnik beschäftigt, beinhaltet die Spieleentwicklung Aspekte anderer Gebiete wie beispielsweise von Medien und Kunst, sodass sich die Frage stellt, an welcher Stelle und in welcher Form sich die Spieleentwicklung als Teilgebiet der Informatik, insbesondere im Verhältnis zur Softwaretechnik, einordnen lässt.

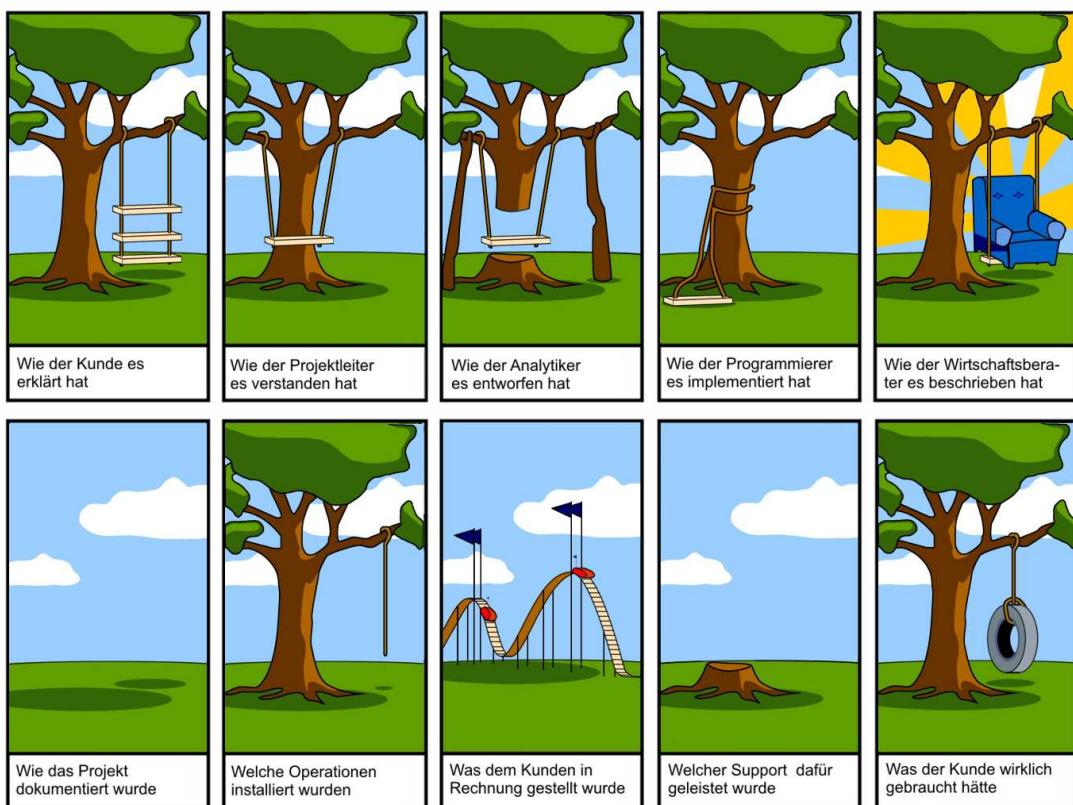


Abbildung 1.2: Illustration von Problemen, mit denen sich Softwaretechnik beschäftigt, Quelle: Paragon Innovations, Inc. (2005)

Mit dem wachsenden Markt und technologischen Weiterentwicklungen steigen auch die Anforderungen an die Spiele und deren Komplexität, zudem müssen Innovationen Spielern einen Anreiz geben, neue Produkte zu kaufen. Neues auszuprobieren stellt ein Risiko für Projekte und Investitionen dar, ebenso ein hoher Komplexitätsgrad vom Spiel oder Entwicklungsprozess, wenn dieser in Planung und Entwurf

nicht angemessen berücksichtigt wird. Bei der Spieleentwicklung sind Wissen und Werkzeuge der Softwaretechnik sehr geeignet, den Entwicklungsprozess zu optimieren. Entwurfsmuster (engl. *Design Patterns*) stellen dabei ein wichtiges Werkzeug dar, um der zunehmenden Größe und Komplexität der Software begegnen zu können ([17], S. 1 bis 5 u. S. 235). Im praktischen Teil der Arbeit, in dem eine Architektur zur automatisierten Spielweltgenerierung entworfen und implementiert wird, wird daher das Augenmerk auf den Entwurf und auf Entwurfsmuster zur Reduktion der Komplexität gerichtet.

Ein solcher Spielweltgenerator ist geeignet, für Spiele mit virtuellen Welten bereits in frühen Phasen des Entwicklungsprozesses spielbare Prototypen zu erstellen, und bietet damit die Möglichkeit, neues auszuprobieren und potentielle Probleme frühzeitig zu erkennen: Durch spielbare Prototypen können Erfahrungen von unterschiedlichen Spielern und somit die Perspektiven verschiedener Arten von Spielertypen (vgl. [4]) in den Game Design-Prozess einfließen. Aufgrund der Vielfalt von Spielwelten und den Anforderungen an sie hängt der Nutzen dieses Tools nicht nur von der Einfachheit und Reduktion der Komplexität auf das Wesentliche ab, sondern auch davon, dass es erweiterbar und anpassbar ist.

1.2 Einführung

Zunächst wird im Theorieteil der aktuelle Stand der Spieleentwicklung auf Basis der Literatur und Wirtschaftspraxis zusammengefasst und der Bereich der Spieleentwicklung definiert. Davon ausgehend werden mögliche theoretische Einordnungen von Softwaretechnik und Spieleentwicklung zueinander formuliert und überprüft.

Im zweiten Teil werden Analyse und Entwurf der Architektur des Spielweltgenerators erst auf hoher Abstraktionsebene geplant und dann *Top-down* modelliert. Trotz der schrittweisen Konkretisierung von Softwarearchitektur und Implementierung wird versucht, die Perspektive des Spielers und des Game-Designers in alle Schritte miteinzubeziehen, um Softwaretechnik in Hinsicht auf spieldspezifische Aspekte wie Ästhetik und Gameplay möglichst effektiv einzusetzen. Anschließend wird die Eignung dieser Methode untersucht, sowie in Hinblick auf den Einsatz von Softwaretechnik in der Spieleentwicklung beurteilt.

Fachbegriffe aus der Spielebranche sind im Glossar im Schlussteil dieser Arbeit aufgeführt und erklärt, sie können dort nachgeschlagen werden.

2 Theorie und verwandte Arbeiten

In diesem Teil werden die Themengebiete der Softwaretechnik und Spieleentwicklung untersucht, um eine Einordnung der Spieleentwicklung in Bezug auf Softwaretechnik vorzunehmen.

2.1 Softwaretechnik

Als Ausgangspunkt einer Einordnung wird zunächst der Begriff der Softwaretechnik eingeführt, indem in diesem Abschnitt Definitionen und Inhalte des Gebiets der Softwaretechnik zusammengefasst werden.

2.1.1 Definition und Abgrenzung

Nach gängigen Definitionen bezeichnet Softwaretechnik (engl. *Software Engineering*) das Teilgebiet der Informatik, das sich mit Methoden und Werkzeugen für die systematische Herstellung, Anwendung und Wartung von Software befasst (vgl. [22], S. 200 bis 213; [42], S. 67).

Zur Softwaretechnik zählen sämtliche Teilschritte des Erstellungsprozesses einer Softwarelösung, der Planung, Analyse, Entwurf, Programmierung und Validierung beinhaltet. Auch weitere Prozesse, die vor allem eine unterstützende Funktion zur qualitativen Verbesserung des Erstellungsprozesses haben wie z. B. Projekt- und Qualitätsmanagement, gehören zur Softwaretechnik.

Die Softwaretechnik ist ein Teilgebiet der Praktischen Informatik. Methoden und Werkzeuge, die für alle Arten von Software gleichermaßen eingesetzt werden, sind der Softwaretechnik zugeordnet.

Weitere Themengebiete der Praktischen Informatik sind Algorithmen, Datenstrukturen, Programmiersprachen, Betriebssysteme und Datenbanken [33]. Die Zugehörigkeit bzw. Zusammenfassung zur Praktischen Informatik ist aus der Eigenschaft abgeleitet, dass diese Teilgebiete der Informatik einen allgemeinen Bezug zu Software haben und vom Abstraktionsgrad zwischen Theoretischer Informatik und speziellen Anwendungsbereichen der Informatik angesiedelt sind.

2.1.2 Bedeutung der Softwaretechnik als Wissenschaftsgebiet

Die Einführung von Methoden und Werkzeugen der Softwaretechnik war nötig, um immer komplexer werdende Softwaresysteme beherrschen zu können. Der Begriff *Software Engineering* wurde erstmals 1968 auf einer Konferenz verwendet ([43], S. 10).

Das grundlegende Prinzip der Softwaretechnik, die Kosten zu minimieren und den Nutzen zu maximieren, wird bei jedem Projekt verfolgt, bei dem ein Unternehmen den Gewinn maximieren möchte. Welche Methoden der Softwaretechnik in welchem Umfang eingesetzt werden, hängt von den speziellen Anforderungen und auch der Anzahl der Beteiligten und der Größe des Projekts ab.

2.2 Spieleentwicklung

Die Entwicklung digitaler Spiele (engl. *Digital Games*¹) beschränkt sich nicht nur auf die Erstellung von Software. Ästhetik, Spielstruktur und Gameplay stellen bei der Charakterisierung dieser Spiele elementare Aspekte dar, die Ebenen und Dimensionen von Modellen zur Einteilung und Beschreibung bilden (vgl. [1], S. 1). Die Sichtweise aus der Informatik, beispielsweise die Ästhetik eines Spiels als nicht-funktionale Eigenschaft eines Softwaresystems zu betrachten, erfasst nur eine von mehreren Dimensionen.

2.2.1 Terminologie

Die Literatur, die sich mit Spieleentwicklung (engl. *Game Development*) beschäftigt, verfolgt bei der Definition der Spieleentwicklung verschiedene Ansätze, die den Fokus und die vorherrschende Perspektive bestimmen, sie lassen sich in die drei Gruppen einteilen:

- Fokussierung auf *Game Design*²: Spieleentwicklung wird beschrieben als Vorgang, der durch das Game Design eingeleitet und bestimmt wird. Die Entwicklung wird vorrangig aus der Sicht eines Game-Designers betrachtet.
- Fokussierung auf den Produktionsprozess: Spieleentwicklung wird ausgehend von Schritten und Bereichen des Produktionsprozesses und den damit verbundenen Tätigkeiten aus konkreten Fallbeispielen und der Praxis in der Spieleindustrie beschrieben.
- Definitionen, bei denen der Softwareentwicklungsprozess und Methoden und Werkzeuge der Softwaretechnik einen Ausgangspunkt darstellen und im Fokus stehen.

¹partiell synonym mit *Video Games*

²auch die Schreibweisen *Gamedesign* und *Game-Design* existieren

Der Begriff Game Design, der Produktionsprozess bei der Spieleentwicklung und weitere Begriffe werden im Folgenden erläutert.

Game Design

Entwurf und Gestaltung eines Spiels (engl. *Game Design*) definieren das Spiel in der Gestalt, wie es sich dem Spieler darstellen soll. Von der ersten Idee an über Konzepte werden primär vor Beginn, aber auch während des Entwicklungsprozesses elementare Eigenschaften eines Spiels bis hin zur detaillierten Ausgestaltung von z. B. grafischen Elementen oder Levels³ festgelegt. Kreative Prozesse, die der Ideensammlung dienen, und eine schrittweise Dokumentation stehen am Anfang und werden Prüfungen unterzogen und verfeinert, bis das zentrale und verbindliche *Game Design Dokument (GDD)* erstellt ist und der Produktionsprozess eingeleitet wird (vgl. [5], S. 101 bis 127).

Produktionsprozess

Die Implementierung der im Game Design Dokument festgelegten Anforderungen bis hin zum fertigen Produkt beinhaltet folgende Aufgaben (vgl. [34], S. 246):

- Softwareentwicklung: Planung, Architektur und Programmierung
- Erstellung von Medien wie Grafik, Sound, Musik und Text
- Tests und Qualitätssicherung

Je nach Art und Umfang des Spiels sind auch folgende Aufgaben Teile des Produktionsprozesses:

- Erstellung von Tools, Demos und Prototypen
- Erstellung von Spielinhalten wie Story⁴, Levels und Scripts
- Risikomanagement
- Wartung und Service

Eine beispielhafte Einteilung in Hauptphasen eines Produktionsprozesses ist in Abbildung 2.1 dargestellt. Die Vorproduktion dient der Planung des Projekts und der Spezifikation von Meilensteinen, Budget und der im Game Design Dokument festgelegten Spieleigenschaften. Dann wird die Entscheidung getroffen, ob die Produktion gestartet wird, in der das Spiel entwickelt wird. In den Phasen Alphatest und Beta-test wird das Spiel zunehmend intern und je nach Projekt auch extern oder öffentlich getestet, um Fehler zu beseitigen und die gewünschte Produktqualität zu erreichen.

³siehe Glossar

⁴siehe Glossar

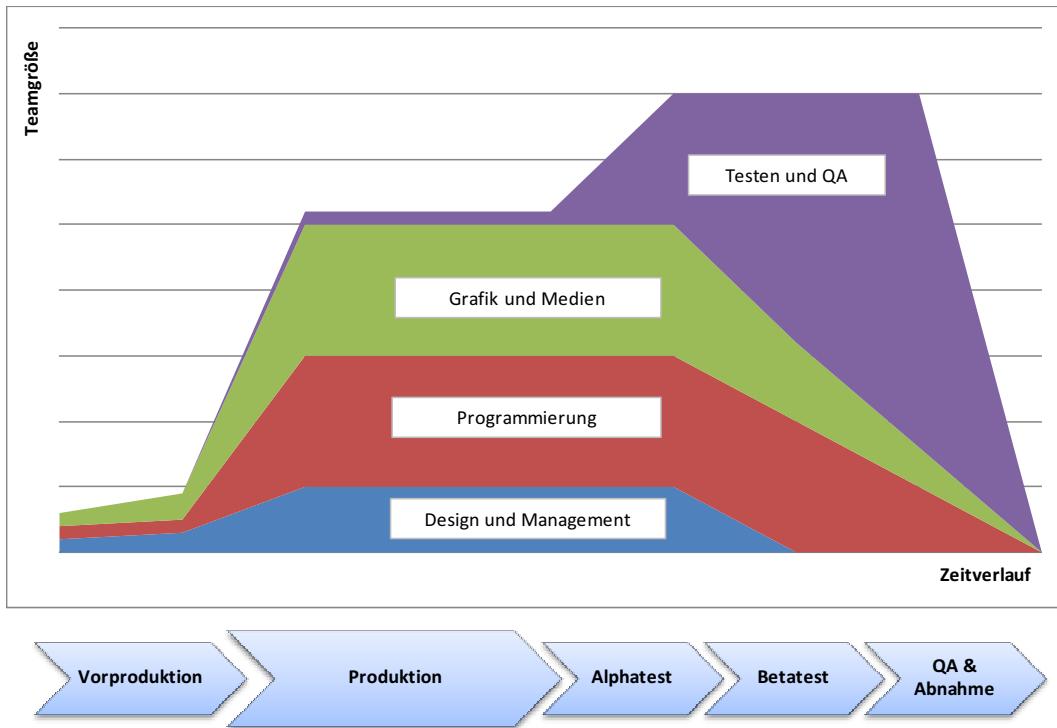


Abbildung 2.1: Produktionsprozess und Entwicklerteam (in Anlehnung an [2], S. 62; vgl. [26], S. 58)

Dieses Modell betrachtet das Spiel als Produkt, welches in der Vorproduktion bzw. durch das Game Design entworfen wird und dann in den einzelnen Phasen implementiert wird. Das Modell ist für viele Projekte typisch, deshalb wird der Spieleentwicklungsprozess von verschiedenen Autoren nach diesem oder einem ähnlichen Modell eingeteilt und beschrieben (vgl. [34], [2]).

Während es allen Projekten gemein ist, dass sich Entwurfs- und Planungsarbeiten vorwiegend in frühen Phasen und Testarbeiten in späten Phasen befinden, ergeben sich bedeutende Unterschiede zwischen Produktionsprozessen durch die Faktoren:

- Profil, Philosophie und Hierarchie des Teams
- Art des Spiels (Genre⁵, Plattform)
- Größe des Teams
- Anteil von Middleware⁶ und Technologien
- Anteil von bereits vorhandenem Produktmaterial

Die Liste lässt sich beliebig erweitern. Unterschiedliche Faktoren ergeben eine andere Gewichtung von Teilprozessen. Während ein größeres Team z. B. die Steuerung

⁵siehe Glossar

⁶siehe Glossar

von Kommunikationsprozessen bzw. Management zu einer umfangreicheren Aufgabe macht, hängt es sehr vom Produkt und der Unternehmenspraxis ab, wie sehr der Prozess auf einzelne Aufgaben ausgerichtet ist. Risikobewertung und Prozessplanung dienen wie bei der Softwaretechnik dem Ziel, Qualität und Effizienz bzw. Effektivität zu maximieren.

Im Gegensatz zu dem linearen Modell aus Abbildung 2.1 lässt sich ein komplexer Entwicklungsprozess auch als *iterativer* und *inkrementeller* Gesamtprozess aus vielen kleineren Teilprozessen beschreiben und umsetzen. Der Gesamtprozess vollzieht sich dann als Folge von teilweise aufeinanderfolgenden und teilweise überlappenden Teilprozessen, die jeweils die Phasen von Design, Architektur, Implementierung und Test durchlaufen (vgl. *V-Modell* [8], und *Spiralmodell* [9], S. 61 bis 72).

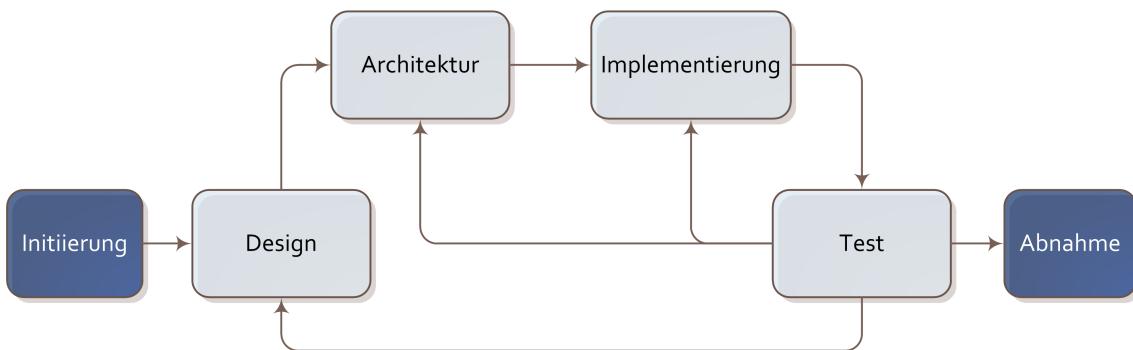


Abbildung 2.2: Teilprozess mit möglichen Zyklen

In Abbildung 2.2 sind die Phasen eines Teilprozesses dargestellt, der die Erstellung eines Produktteils wie z. B. eines Spielinhalts (Level⁷, Grafik oder sonstige Medien), eines Softwaremoduls oder eines Prototyps beschreibt. Der Test des Produktteils kann außer der erfolgreichen Abnahme auch zur Folge haben, dass:

- Änderungen an der Implementierung, der Architektur oder dem Design vorgenommen werden müssen, sodass der Teilprozess aus mehreren Zyklen besteht.
- Ein anderer, neuer Teilprozess initiiert werden muss, um aufgetretene Probleme zu lösen.

Ein solcher Teilprozess kann sich zeitlich komplett in der Vorproduktion befinden, weil die Produktion erst nach erfolgreichem Abschluss gestartet werden kann. Es ist üblich, im Vorfeld einer Spielproduktion bereits einen Prototyp fertigzustellen, um eine Grundlage für fundierte Kosten- und Ressourcenplanungen zu haben, oder zur Akquisition von Finanzmittelgebern ([26], S. 60). Somit fallen bereits in der Vorproduktionsphase auch Implementierungs- und Testarbeiten an.

⁷siehe Glossar

Ästhetik, Gameplay und Spielmechanik

Der Begriff *Gameplay* wird in der Literatur mit voneinander stark abweichenden Bedeutungen verwendet⁸. Die in dieser Arbeit verwendeten Bedeutungen von Ästhetik, *Gameplay* und Spielmechanik als Eigenschaften digitaler Spiele werden hier erläutert.

Die *Ästhetik* bezeichnet alle Eigenschaften eines Spiels, die der Spieler wahrnimmt und wahrnehmen kann. Aufgrund der Subjektivität der Wahrnehmung jedes einzelnen umfasst die Ästhetik auch beispielsweise Emotionen, die ein Spiel transportiert und Faktoren, die weder sichtbar noch technisch erfassbar, jedoch wahrnehmbar sind. Das Begriffsfeld ist im Gegensatz zur Alltagssprache, die etwas Schönes als ästhetisch bezeichnet, allgemein und wertneutral.

Gameplay wird zur Bezeichnung eines engeren Begriffsfeldes als Ästhetik verwendet: Es bezeichnet die Eigenschaften des Spiels und Erfahrungen des Spielers, die einen direkten Bezug zur Interaktion des Spielers mit dem Spiel oder dem aus Spielregeln und Interaktion entstehenden Spielablauf haben.

Spielmechanik bezeichnet die durch die Spielregeln eines Spiels bestimmten Spielabläufe, die ein Spiel ermöglicht. Abbildung 2.3 zeigt die drei Bedeutungsfelder.



Abbildung 2.3: Ästhetik, *Gameplay* und Spielmechanik

2.2.2 Definition und Abgrenzung

Mit Spieleentwicklung wird in dieser Arbeit die Entwicklung von digitalen Spielen, d. h. Spielen in virtuellen Umgebungen (vgl. [1], S. 1), bezeichnet. Der Begriff Spieleentwickler (engl. *Game Developer*) ist für Firmen und als Berufsgruppe üblich.

Der Themenbereich der Spieleentwicklung lässt sich daher definieren und abgrenzen durch:

- die tatsächlichen Tätigkeitsbereiche von Spieleentwicklern (wie im Abschnitt *Produktionsprozess* aufgeführt)

⁸teilweise synonym mit Spielmechanik und teilweise synonym mit Ästhetik verwendet

- wissenschaftliche Fragestellungen und Forschungsthemen, die sich auf die Entwicklung von digitalen Spielen beziehen (Game Design und auch theoretische Aspekte zu digitalen Spielen, die einen Bezug zur Entwicklung haben)

Die sich daraus ergebenden größeren Themengebiete der Spieleentwicklung sind:

- Softwareentwicklung
- Game Design
- Mediendesign⁹

Softwareentwicklung lässt sich der Softwaretechnik und der Informatik zuordnen, Mediendesign den Kunstwissenschaften. Game Design beschäftigt sich vorwiegend mit den Aspekten, die für Spiele spezifisch sind: Ästhetik, Spielstruktur und Gameplay.

J. Bragge und J. Storgårds zeigen in ihrer Studie einen hohen Grad an Multidisziplinarität der akademischen Literatur zu digitalen Spielen und Spieleentwicklung auf ([10], S. 714).

Für eine wissenschaftliche Sicht auf die Spieleentwicklung mit den technischen, künstlerischen und spielspezifischen Aspekten darf deshalb keine Wissenschaftsdisziplin im Vorfeld ausgegrenzt werden, sondern die Multidisziplinarität muss berücksichtigt werden.

2.2.3 Andere Definitionen

Es existieren auch Definitionen der Spieleentwicklung, die andere Ausgangspunkte wählen und verfolgen, die wichtigsten werden hier genannt, um den Rahmen aufzuzeigen, in dem sie sich bewegen.

Spieleentwicklung als Softwareentwicklung

E. Bethke klassifiziert Spieleentwicklung als Softwareentwicklung und schreibt, dass Spieleentwicklung Softwareentwicklung ist, weil Spiele Software mit Grafik, Audio und Gameplay sind:

Games are certainly special; however [...] game development is software development. Games are software with art, audio, and gameplay. Financial planning software is software that is specialized for financial transactions and planning [...]. Too often game developers hold themselves apart from formal software development and production methods with the false rationalization that games are an art, not a science. Game developers need to master their production methods so that they can produce their games in an organized, repeatable manner [...]. ([5], S. 4)

⁹auch die Schreibweisen Mediadesign und Medien-Design existieren

Das Zitat zeigt, dass die Aussage „Spieleentwicklung ist Softwareentwicklung“ nicht zwingend einen Widerspruch zur Multidisziplinariät der Spieleentwicklung darstellt, sondern ein Argument für den Einsatz von Methoden der Softwaretechnik in der Spieleentwicklung liefert.

Es stellt sich die Frage, ob die Aussage, dass Spiele Software mit Grafik, Audio und Gameplay sind, nach wissenschaftlichen Maßstäben dafür geeignet ist, auf Basis dieser Klassifikation eine allgemeine und objektive Definition für die Spieleentwicklung abzuleiten. Dass in der Spieleentwicklung Software entwickelt wird, lässt sich ebenso wenig bestreiten, wie dass Medien und Gameplay hergestellt werden. Alle sind wichtige Bestandteile des Spiels.

Game Design als übergeordnetes Element der Spieleentwicklung

Mehrere Bücher zum Thema Game Design beschäftigen sich nicht nur mit Entwurfsphasen, sondern der gesamten Entwicklung von Spielen (vgl. [18], [19], [35]). Das Begriffsfeld des Wortes *Design* wird dort umfassender verstanden als im Kontext von Softwareentwicklung bzw. Softwaretechnik, es bezieht sich dort insbesondere nicht nur auf Planung und Entwurf, d. h. auf die theoretische Konzeption, sondern auch auf konkrete gestalterische Aspekte der Ästhetik des Spiels und die praktische Definition des Gameplay.

J. Schell verdeutlicht dies, indem er Game Design explizit nicht auf den Tätigkeitsbereich bestimmter Personen beschränkt:

Anyone who makes decisions about how the game should be is a game designer. Designer is a role, not a person. ([36], S. XXIV)

Für die Spieleentwicklung bedeutet dies, dass das Game Design sowohl bei der theoretischen Konzeption als auch bei der praktischen Umsetzung aktiv die Entwicklung bestimmt und sich übergeordnete Phasen und Struktur des Entwicklungsprozesses am Game Design, und nicht an der Softwareentwicklung orientieren: Die Softwareentwicklung stellt als Umsetzung der einzelnen Phasen Teilprozesse dar.

2.3 Softwaretechnik und Spieleentwicklung

Da sich Softwaretechnik mit der Softwareentwicklung beschäftigt, unabhängig von der Frage, um welches Anwendungsgebiet es sich bei der Software handelt, lässt sich die Spieleentwicklung als Teilgebiet der Softwaretechnik sehen: Es wird eine spezielle Softwareart, nämlich digitale Spiele entwickelt.

Die Spieleentwicklung lässt sich gleichzeitig auch als Themengebiet betrachten, von der die Software bzw. deren Entwicklung nur einen Teil darstellt: Vor allem die Idee des Spiels und die Spielregeln sind nicht Teil der Softwareentwicklung, sondern werden durch diese umgesetzt.

Argumente für beide Positionen werden in diesem Abschnitt analysiert und eine Einordnung von Softwaretechnik und Spieleentwicklung vorgenommen.

2.3.1 Spiel und Software

Beide Positionen haben eine unterschiedliche Definition des Spiels, welches Gegenstand der Entwicklung ist, zum Ausgangspunkt.

Das Spiel als Software

Wird die Softwareeigenschaft als wichtigstes Merkmal des digitalen Spiels gesehen und das Spiel als „Software mit Grafik, Audio und Gameplay“ klassifiziert, ergibt sich daraus, dass Spieleentwicklung die Entwicklung einer speziellen Software, des Spiels, ist.

Es ist Aufgabe und Ziel des Softwareentwicklers, die Software möglichst effizient und qualitativ hochwertig zu entwickeln, und die Softwaretechnik bietet genau Methoden und Werkzeuge dafür.

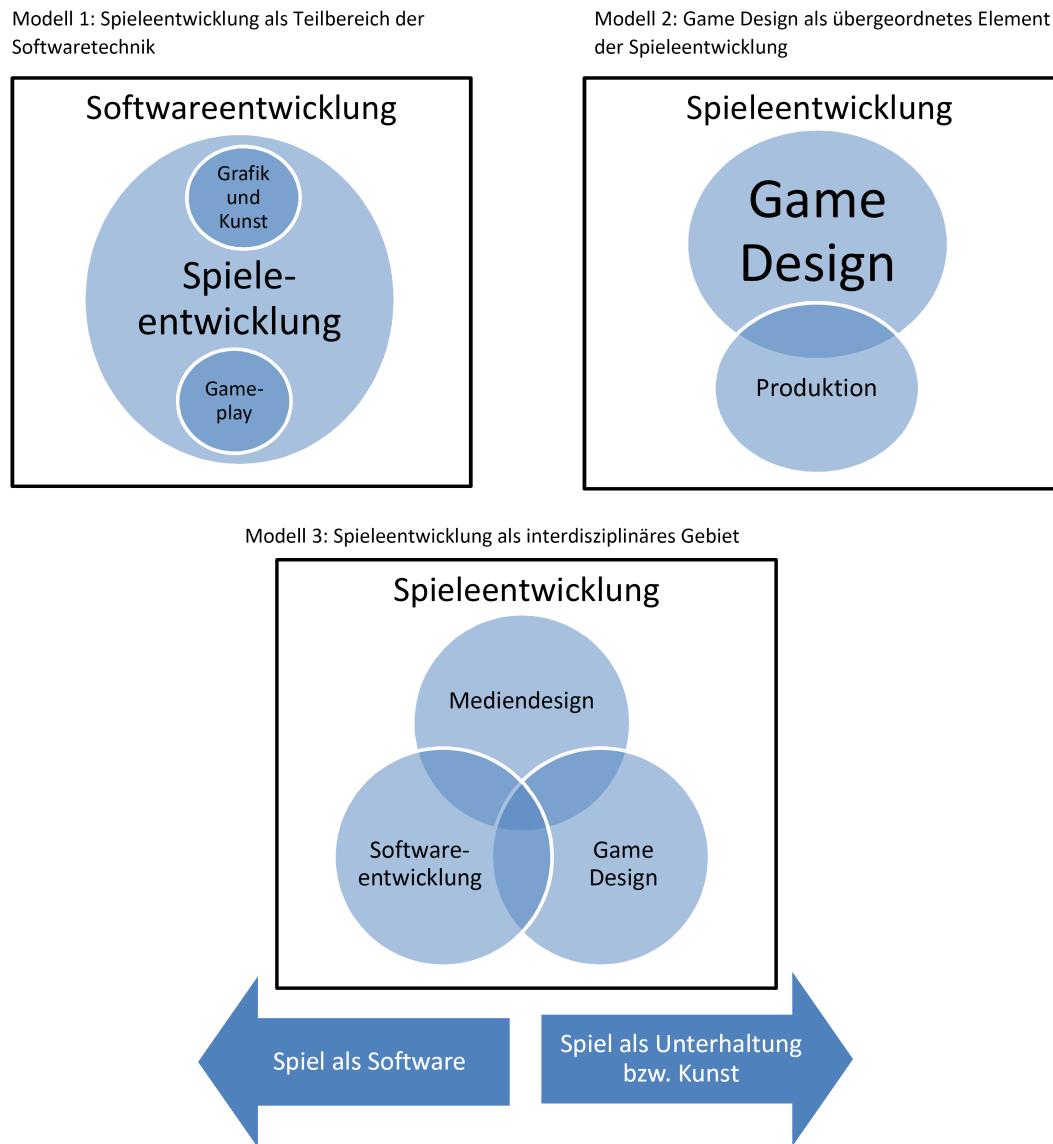
Das Spiel als Unterhaltung

Betrachtet man das Spiel als eine Tätigkeit der Unterhaltung, die nach den Spielregeln mit dem Gameplay und der Ästhetik des jeweiligen digitalen Spiels abläuft, so ist die Spielsoftware die in Form von Programmquelltext und Medien materialisierte Spielidee, die durch das Game Design entworfen wird.

Wer bei der Spieleentwicklung am Game Design mitwirkt, aber selbst keine Software entwickelt, sondern die Anforderungen des Spiels kreiert, dessen Aufgabe und Ziel ist es vielmehr, ein gutes und innovatives Spiel mit hohem Unterhaltungswert und interessantem Gameplay zu entwerfen, als sich mit Softwareaspekten zu beschäftigen. Diese Aspekte beziehen sich ausschließlich auf die äußere, wahrnehmbare Spielform, auf Aspekte des Spiels als Tätigkeit der Unterhaltung. Wie die Software möglichst effizient die Spielidee umsetzt, ist dann nicht Aufgabenbereich des Game-Designers, sondern des Softwareentwicklers.

Vergleich der Modelle

Abbildung 2.4 gibt eine Übersicht über drei verschiedene Modelle zur Definition der Spieleentwicklung.

**Abbildung 2.4:** Verschiedene Modelle

Argumente und Widersprüche

Die Modelle sind aus der durch die jeweilige Perspektive bedingten Definition eines Spiels hergeleitet. Um die Frage nach einer möglichst objektiven Einordnung und nicht mehreren subjektiven und widersprüchlichen Klassifizierungen, beantworten zu können, sind in Tabelle 2.1 die Hauptargumente für die unterschiedlichen Positionen zusammengefasst und gegenübergestellt.

Argumente für Modell 1: Spieleentwicklung als Teilbereich der Softwaretechnik	Argumente für Modell 2: Game Design als übergeordnetes Element der Spieleentwicklung
Softwaretechnik beschäftigt sich mit allen Aspekten der Software und ist nicht auf eine bestimmte Art von Software beschränkt (vgl. [43], S. 31 bis 33). Digitale Spiele sind Software mit Medien und Gameplay, daher ist Spieleentwicklung ein spezieller Bereich der Softwareentwicklung (vgl. [5], S. 4).	Game Design wird stark vom Objekt des Designs, den Spielen, bestimmt und findet während der gesamten Entwicklung statt (vgl. [23], S. 5 und S. 7). Prototyping von Spielen ist die Möglichkeit, Ideen auszuprobieren und bezieht sich auf das Spiel und nicht auf Softwaretechnik, die Software ist nur ein Teil vom Spiel (vgl. [19], S. 219). Eigenständige Methoden und Muster für das Game Design werden gefordert (vgl. [29], [7]).
Probleme der Softwareentwicklung zählen zu den größten Risiken von Spieleentwicklungen. Spiele- und Softwareprojekte verzögern sich sehr oft (vgl. [5], S. 41). Methoden und Werkzeuge der Softwaretechnik stellen eine erprobte und geeignete Lösung dafür dar.	Formale Methoden des Software Engineering funktionieren nur, wenn man von Beginn an exakt weiß, wie das Spiel definiert ist und wie es am Ende aussehen wird (vgl. [34], S. 185).
Agile und iterative Prozessmodelle bieten die Flexibilität, sodass sie mit dem Game Design-Prozess kombiniert werden können (vgl. [34], S. 185).	Traditionelle Prozessmodelle können Kreativität und Innovation des Game Designs einschränken und dazu führen, das eigentliche Ziel und den Nutzen des Spiels, aus den Augen zu verlieren (vgl. [36], S. 405).
Game Design definiert nur abstrakt die Spielidee mit Erwartungen an Spielspaß und Innovation. Die Umsetzung ist von den praktischen Möglichkeiten abhängig und durch diese beschränkt, bestimmte Arten von Gameplay sind von der Technologie abhängig (vgl. [36], S. 3, S. 77 und S. 409; [5] S. 224). Komplexe Technologie und Software ist durch Softwaretechnik professionell beherrschbar.	Sehr wichtige Anforderungen an ein Spiel, wie Spaß bzw. emotionale Faktoren, werden von der Software-Anforderungsanalyse selbst als nichtfunktionale Anforderungen kaum bis gar nicht erfasst, da sie nicht Teil der Software sind (vgl. [11], S.2).

Tabelle 2.1: Gegenüberstellung von Argumenten

Die Argumente sprechen mehrheitlich für einen Einsatz von Werkzeugen und Methoden der Softwaretechnik innerhalb der Softwareentwicklung und gegen eine negative Beeinflussung des Game Designs.

Werkzeuge und Methoden der Softwaretechnik wie z. B. strukturierte Prozessmodelle und Risikomanagement sind teilweise nicht auf softwarespezifische Aspekte beschränkt und geben auch in Bezug auf den gesamten Entwicklungsprozess eine Antwort auf die Frage, *wie* das Spiel technisch bestmöglich entwickelt werden kann, auch aufgrund der Nähe der Softwaretechnik zu Ingenieurwissenschaften und anderen Gebieten.

2.3.2 Nutzenmaßstab der Spieleentwicklung

Der Maßstab für Nutzen und Qualität der Spieleentwicklung lässt sich durch Effizienz und Effektivität beschreiben, d. h. das Ziel einer professionellen Spieleentwicklung sollte zum einen geringe Kosten verursachen und zum anderen ein gutes Spiel produzieren.

Die Frage, wann ein Spiel gut ist, und welches die wichtigsten Eigenschaften sind, ist sehr umstritten. So wird an vielen Stellen Spielspaß bzw. Gameplay als der wichtigste Aspekt genannt (vgl. [7], S. 3). Abbildung 2.5 listet wichtige Merkmale eines guten Spiels im Vergleich zu denen von guter Software auf: Der Spielspaß, der sich aufgrund der Ästhetik, des Gameplay und der Spielstruktur ergibt, ist für den Spieler ein wichtiges Kriterium. Den Spielspaß zu erklären, ist Forschungsgebiet und hängt von subjektiven Faktoren ab, verschiedene Menschen erleben in unterschiedlichen Phasen von Spielen höchst unterschiedliche Quantitäten und Qualitäten von Spielspaß ([28], S. 12 bis 13). Trotz der Subjektivität stellt Spielspaß den Grund und Zweck dar, weshalb ein Spiel überhaupt gespielt wird. Anders als bei den Merkmalen guter Software, wo Grund und Zweck ausgegrenzt bzw. als Konstante angenommen werden, stellt er bei Spielen ein notwendiges Kriterium für die Güte des Spiels dar. Denn ein Spiel, welches dem Spieler keinen Spaß bereitet, wird dieser nicht nur deshalb spielen, weil es alle Merkmale guter Software erfüllt.

Auch die Innovation ist ein wichtiges Merkmal, das als Grund und Zweck einer Spieleentwicklung betrachtet werden muss: Bei Spielen mit bewährten und bekannten Spielregeln, wie z. B. Schach oder Poker, die schon in vielen Umsetzungen existieren, sowie bei Folgeversionen von Produkten, müssen Innovationen (vgl. *Unique Selling Points* [34], S. 17) dafür sorgen, dass diese Spiele erneut produziert, gekauft und gespielt werden. Innovationen sind für den Käufer und Spieler ein wichtiges Beurteilungskriterium der Spiele.

2.3.3 Einordnung von Softwaretechnik und Spieleentwicklung

Anhand der Perspektiven, Modelle und des Nutzenmaßstabs wird nun die Einordnung von Softwaretechnik und Spieleentwicklung vorgenommen.

Bei der Spieleentwicklung lassen sich nahezu alle Methoden und Werkzeuge der Softwaretechnik einsetzen, um die Spieleentwicklung zu professionalisieren und Risiken

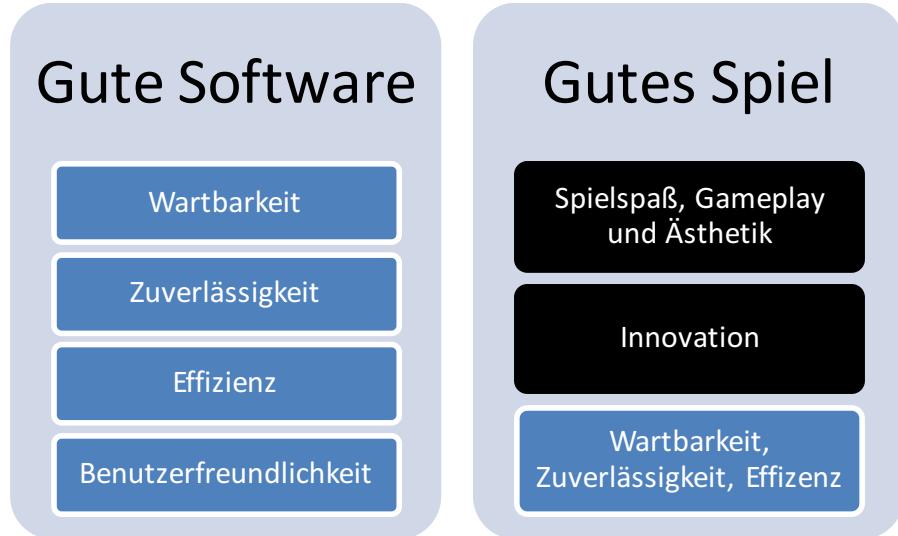


Abbildung 2.5: Vergleich von Software und Spiel (vgl. [43], S. 39)

zu minimieren (vgl. [17], S. 1 bis 26). Dies gilt nicht nur im Rahmen der Softwareentwicklung, sondern auch für den gesamten Entwicklungsprozess können Methoden und Werkzeuge der Softwaretechnik benutzt oder übertragen werden.

Die Spieleentwicklung stellt ein spezielles Anwendungsgebiet der Softwaretechnik dar, bei dem Software von digitalen Spielen entwickelt wird. Softwaretechnik besitzt daher bezüglich der Methoden und Werkzeuge für Softwaresysteme und Entwicklungsprozesse ein größeres und in Bezug auf das Anwendungsgebiet der Software abstrakteres Themenfeld.

Umgekehrt bestehen Spiele nicht nur aus Software: Game Design bildet mit den Maßstäben Spielspaß, Gameplay und Ästhetik eine andere Priorisierung und ein anderes Wissenschaftsgebiet, welches ein umfassenderes und abstrakteres Themenfeld umfasst, als das vom ingenieurmäßigen Software- und Systementwurf: Spielspaß lässt sich eher „erfühlen“ als mit klaren Begriffen zu definieren. Zur genaueren Beschreibung verwendete Begriffe wie Erfolgserlebnis und Spannung, oder Emotionen wie Stolz, Neugier, Ungewissheit, Pflichtgefühl, Rachsucht, Schadenfreude und Ordnungsliebe (vgl. [37], S. 25 bis 27) sind *subjektive Erlebnisse*, die sich nicht als technische Größe definieren und beziffern lassen, sondern nur durch Hilfsgrößen und eigene Modelle erfassen und kommunizieren lassen (vgl. [12], S. 299 bis 302).

Der Spielspaß ist notwendige und übergeordnete Anforderung an alle Spiele und wird aus Sicht des Spielers und Game-Designers im Game Design primär nach Aspekten definiert, die unabhängig von Software, Technik und Systemen bestehen: Der Spielspaß bezieht sich nicht auf Aspekte des Softwaresystems, denn diese sind für den Spieler nicht sichtbar (vgl. [17], S. 13).

Da in der Spieleentwicklung oft in multidisziplinären Teams zusammengearbeitet wird und somit verschiedene Arten von *Stakeholdern*, d. h. Beteiligten mit einem

berechtigten Interesse am Verlauf und Ergebnis des Gesamtprozesses, involviert sind, sollte das Verständnis der unterschiedlichen Perspektiven und eine einheitliche, interdisziplinäre Sicht auf den gesamten Spieleentwicklungsprozess im Vordergrund stehen (vgl. [23], S. 2).

Dazu ist es hilfreich, die Unterschiede der Perspektiven und die Schnittstelle und Kommunikation zwischen Game Design und Softwareentwicklung genauer zu betrachten.

Beim Übergang von der Vorproduktionsphase zur Produktionsphase ist das Game Design zu einem bestimmten Teil abgeschlossen und die durch das Game Design Dokument weniger formal festgelegten Anforderungen gehen in eine technische Softwarespezifikation über. Dieser Vorgang ist besonders schwierig, da an dieser Schnittstelle zwischen Game Design und Softwareentwicklung Faktoren von künstlerischer und spielspezifischer Natur formalisiert werden müssen (vgl. [11], S. 4). Im Game Design Dokument enthaltene implizite Informationen können bei unterschiedlichen Perspektiven von Game-Designer und Softwareentwickler nicht eindeutig auf die technische Softwarespezifikation abgebildet werden, sodass von E. Callele et al. formale Methoden zur Bewerkstelligung dieses Übergangs gefordert werden ([11], S. 9). In diesem Abschnitt des Produktionsprozesses haben die Anforderungsanalyse und die Kommunikation zwischen Game-Designer und Softwareentwickler besondere Wichtigkeit für das gesamte Projekt, da ab diesem Zeitpunkt von allen Beteiligten die meisten Ressourcen eingesetzt werden.

Eine Klassifizierung der Spieleentwicklung als Teilbereich der Softwaretechnik würde die Relevanz dieser speziellen Schnittstelle von Game Design und Softwareentwicklung im Spieleentwicklungsprozess eher ausblenden, statt diese angemessen zu berücksichtigen.

Eine objektive Grundlage für ein Verständnis der unterschiedlichen Perspektiven und eine einheitliche Kommunikation stellt die interdisziplinäre Sicht auf die Spieleentwicklung dar, da sie die Perspektiven aller *Stakeholder* miteinbezieht. Aus Sicht der Softwaretechnik muss daher Game Design als ein externes Konzept verstanden werden: Methoden und Werkzeuge der Softwaretechnik können in der Spieleentwicklung am effizientesten unter Rücksichtnahme auf die Besonderheiten der Softwareentwicklung im Kontext des Game Designs, wie des speziellen Nutzenmaßstabs von Spielen, eingesetzt werden (vgl. [17], S. 1 bis 3).

In Abbildung 2.6 sind die an die Spieleentwicklung angepassten Methoden und Werkzeuge der Softwaretechnik als *Game Engineering* verdeutlicht: Zur systematischen Verbesserung des Spieleentwicklungsprozesses ist das *Game Engineering* nicht auf das Produkt Software, sondern auf das Produkt Spiel ausgerichtet, welches sich einerseits als Software und andererseits als vom Game Design definiertes Unterhaltungsmedium versteht. Daher berücksichtigt *Game Engineering* sowohl Softwareentwicklung als auch Game Design, um den gesamten Spieleentwicklungsprozess zu optimieren.

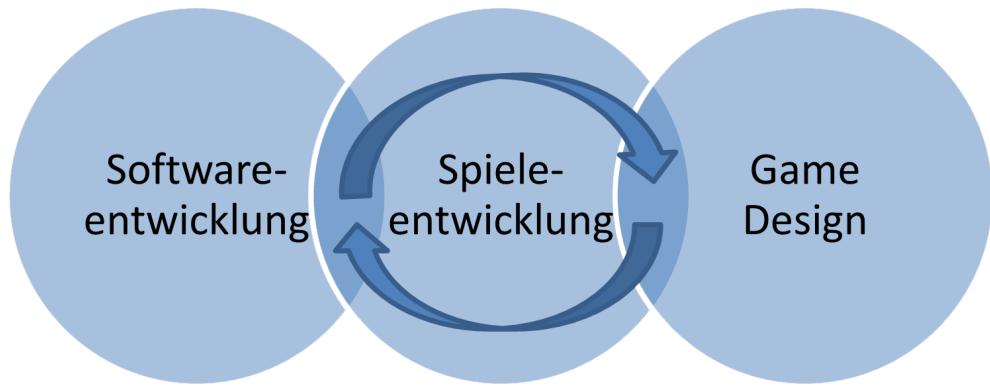


Abbildung 2.6: Game Engineering in der Spieleentwicklung

2.4 Fazit

Ästhetik und Gameplay von digitalen Spielen werden durch das Game Design bestimmt. Die Perspektive des Game-Designers und die spielspezifischen Ziele Spielspaß und Innovation sollten im Entwurf und Softwareentwicklungsprozess besondere Berücksichtigung finden und in der Anforderungsanalyse erfasst werden. Agile und iterative Entwicklungsprozesse helfen, Aspekte des Game Designs besser in die Softwareentwicklung integrieren zu können und somit die Spieleentwicklung in höherem Maß vom Einsatz der Methoden und Werkzeuge der Softwaretechnik profitieren zu lassen.

Durch eine interdisziplinäre Sichtweise auf die Spieleentwicklung wird eine ganzheitliche Risikobewertung und damit ein geeignetes Risikomanagement möglich, das Risiken nach Höhe und Ausmaß, die sie für das Projekt bzw. Produkt darstellen, beurteilt und somit weniger Gefahr läuft, die Risiken aufgrund einer einseitigen Perspektive nicht angemessen zu bewerten.

3 Projekt Spielweltgenerator

In diesem Teil werden Modell, Architektur und Implementierung einer Software zur automatisierten Generierung von Spielwelten beschrieben.

3.1 Ziel und Methodik

Dieser Abschnitt stellt die verfolgten Ziele dar und erläutert die gewählte Vorgehensweise. Es werden auch Probleme genannt und die Schwerpunkte des Projekts eingegrenzt.

3.1.1 Zielsetzung

Im Projekt wird das Ziel verfolgt, eine Software zur automatisierten Generierung einer 3D-Spielwelt zu erstellen, sodass das generierte 3D-Modell direkt für Prototypen von Computerspielen verwendet werden kann. In Abbildung 3.1 ist die Funktion der Software illustriert. Über die Bedienschnittstelle sollen nur essentielle Daten, die die Spielwelt beschreiben, in das System eingegeben werden müssen und die Architektur der Software muss kompakt und ausbaufähig sein, sodass Änderungen und Erweiterungen an Programmteilen vorgenommen werden können, ohne dass ungewünschte Nebeneffekte auftreten.



Abbildung 3.1: Funktion Spielweltgenerator

3.1.2 Nutzen

Ein zu einem frühen Zeitpunkt des Produktionsprozesses erstellter spielbarer Prototyp bietet die Vorteile:

- Ästhetik und vor allem Gameplay, wichtige Faktoren für den Spielspaß, können getestet und modifiziert werden.
- Spielspaß und Marktpotenzial eines Spiels können abgeschätzt und Geldgeber akquiriert werden.
- Kosten für Präsentationen können minimiert werden: Ein spielbarer Prototyp besitzt mehr bzw. zusätzliche Aussagekraft zu *Artwork*¹ und Dokumentationen.
- Risiken und Probleme, die sonst erst zu einem späteren Zeitpunkt auftreten, werden früher sichtbar und können besser beseitigt werden. Das *Balancing*¹ beispielsweise kann ab einem bestimmten Komplexitätsgrad nicht exakt berechnet werden, sondern wird erst durch Tests und Simulationen entscheidend angepasst (vgl. [15], S. 31).

Bei vielen verkaufsstarken Genres, seien es nun Simulationen, Rollenspiele¹, Ego-Shooter¹ oder andere, befindet sich der Spieler in einer 3D-Spielwelt. Während in einigen Fällen direkt Fotorealismus und physikalischer Realismus angestrebt wird, variiert es je nach Spiel, inwiefern das Gameplay und der Spielspaß von der 3D-Darstellung beeinflusst werden. Da z. B. bei einem Schachspiel die möglichen Spielzüge, die den Spielverlauf bestimmen, aufgrund der determinierten Regeln jedem Spieler verständlich sind, besteht der Spielspaß weitgehend unabhängig von der Darstellung. Bei Spielen, die sich im 3D-Raum abspielen und eine Interaktion mit Objekten beinhalten, wirkt sich die Räumlichkeit direkt auf das Gameplay und das Immersionspotenzial eines Spiels aus (vgl. [21]).

3.1.3 Abgrenzung und Kompromisse

Um den Aufwand in Grenzen zu halten, werden Umfang und Performance der zu entwickelnden Software in der Planung an folgenden Stellen beschränkt:

- Die Drahtgitter-Repräsentation der Spielwelt wird so einfach wie möglich gestaltet und nur zu einem für Prototypen hinreichenden Detailgrad.
- Renderqualität², Grafikeffekte und Performance werden ebenfalls nur elementar implementiert und optimiert.
- Die Spielwelt wird auf eine überschaubare Zahl von Landschaftsobjekten wie z. B. Pflanzen, Wasser, Steinen reduziert.

Diese Einschränkungen ermöglichen, dass die im Projekt entwickelte Software die Spielwelt bereits während der Erstellung und Modifizierung realistisch in 3D darstellen kann. Die Spielwelt kann im weiteren Entwicklungsprozess durch eine professionelle Grafikengine in höherer Qualität dargestellt werden und die einfache

¹siehe Glossar

²siehe Glossar

Drahtgitter-Repräsentation durch höherrauflösende 3D-Modelle verbessert werden, sodass generierte Spielwelten nicht nur für Prototypen, sondern auch für darauf aufbauende Produktionen verwendet werden können.

3.1.4 Stand von Industrie und Wissenschaft

Es gibt je nach Anwendungsbereich Tools wie Terraingeneratoren, Grafikengines, 3D-Editoren und Techniken zur prozeduralen Synthese (vgl. [39]), die jedoch entweder nur sehr grobe Spielwelten bzw. Landschaften generieren können, oder sich auf unterster Komplexitätsebene im Detailbereich befinden (vgl. [40], S. 2 und 4).

Auf Techniken zur prozeduralen Generierung von Terrain (vgl. [31]) wird in dieser Arbeit zurückgegriffen, aber nicht im Detail eingegangen, da sich die Arbeit mit einer Architektur und dem Generierungsprozess auf hoher Abstraktionsstufe beschäftigt. Ziel ist, diese Techniken zu kombinieren und in der Spieleentwicklung in höherem Maße automatisiert einsetzen zu können (vgl. [38]).

3.1.5 Methodik

Um Software kostengünstig weiterentwickeln zu können, ist es nützlich, die Änderungen, ihre Eigenschaften und Kosten vorhersagen zu können. So ist es möglich, zur Senkung von künftigen Wartungskosten besonders komplexe Komponenten durch einfachere Alternativen zu ersetzen ([25], S. 335 bis 343). Ein Architekturdesign kann darüber hinaus die Komplexität und Natur von künftigen Änderungen auch in der Modularisierung und Gewichtung von Entwurfsprinzipien widerspiegeln. Dies wird beim Entwurf der Architektur besonders verfolgt, sodass der Änderbarkeit und Erweiterbarkeit der Spielwelt ein besonderes Gewicht zugebilligt wird.

Eine gute Spielwelt richtet sich nach dem Charakter des jeweiligen Spiels. Für das Modell einer universell einsetzbaren bzw. erweiterbaren Spielwelt gelten daher Anforderungen, die weder qualitativ noch quantitativ direkt auf bestimmte Softwarekomponenten abgebildet werden können, denn das Game Design, welches auch während des Entwicklungsprozesses stattfindet ([23], S. 5), soll keinen oder möglichst wenigen Zwängen, die sich aus der Softwareumsetzung ergeben, unterliegen. Daher werden Prinzipien der Softwaretechnik auf möglichst hoher Abstraktionsstufe angewandt, sodass eine Architektur, die eine flexible Spielweltmodellierung begünstigt, angestrebt wird. Daher stützt sich der Architekturentwurf zur Reduktion der Softwarekomplexität auf die Kombination von Entwurfs- bzw. Architekturmustern und fügt sich nicht strikt in ein traditionelles Architekturschema ein.

3.1.6 Lösungsansatz

Die Schwierigkeit der Aufgabenstellung besteht darin, die quantitativ und qualitativ verlangte Flexibilität der Spielwelt, die vom Charakter des Spiels bzw. Gameplay

und der Ästhetik bestimmt werden kann, bei möglichst geringer Softwarekomplexität und einfacher Bedienbarkeit zu realisieren.

Bei der Frage, wie die erstellte Spielwelt im Detail aussehen soll und welche Faktoren einstellbar sein sollen, führen erste Konkretisierungen zu einer Vielzahl von Aspekten, die miteinander gekoppelt werden müssen. Wegen der Vielfalt von unterschiedlichen Spiele und deren Spielwelten dienen realistische, virtuelle Welten und die Realität als Vorbild für das abstrakte Modell, welches im Folgenden entworfen wird.

Die gröbsten und zuerst wahrnehmbaren Elemente einer solchen Welt sind Erde und Wasser. Im 3D-Raum bestimmen sie elementar das Auftreten und die Eigenschaften von anderen sichtbaren Elementen wie Lebewesen, vom Menschen Errichtetes und auch Naturerscheinungen. Im Folgenden werden diese Phänomene als *Objekte* bezeichnet. Feuer und Luft sind im Zusammenhang mit virtuellen Welten Grafikeffekte und werden deshalb auf dieser Abstraktionsebene nicht miteinbezogen.

Das grobe Modell fasst die Welt als Gebilde aus Erde und Wasser auf, von dem die Oberfläche sichtbar ist. Die Oberfläche selbst weist in Farbe und Form verschiedene Eigenschaften wie eine bergige oder flache Struktur und Materialeigenschaften auf. Hauptsächlich auf der Oberfläche befinden sich die Pflanzen und andere sichtbare Objekte, die einfach nur sichtbar sind oder auch durch Kollision oder in anderer Weise das Spiel beeinflussen bzw. mit anderen Objekten interagieren können.

Die Hauptfaktoren, die allgemein bestimmen, welches Objekt sich an einer bestimmten Stelle befindet und bestimmte Eigenschaften aufweist, sind:

- dass es *realistisch* ist, d. h. dass eine Notwendigkeit oder Widerspruchsfreiheit des abgebildeten Zustands zu der realen Welt besteht
- dass der abgebildete Zustand eine bestimmte *Funktion* für das jeweilige Spiel erfüllt und deshalb absichtlich gesetzt wird

Eine Pflanze, die viel Feuchtigkeit benötigt, kann z. B. nicht in der Wüste wachsen, sondern muss sich in einem geeigneten klimatischen Umfeld befinden. In Abbildung 3.2 wird dies anhand eines Beispiels dargestellt: Links sind die Bäume wahllos verteilt, sodass sie sich auf beliebigem Untergrund und auch im Wasser befinden, im rechten Bild befinden sie sich nur auf bestimmtem Untergrund und nicht im Wasser, was realistischer erscheint.

Andere Objekte und Phänomene dürfen sich auch nicht an jedem beliebigen Ort befinden: Straßen und Wege verbinden z. B. Siedlungen bzw. müssen i. A. Teil eines Wegenetzes sein. Diese Beschränkungen lassen sich durch Bedingungen modellieren, anhand derer ein gültiger räumlicher Bereich berechnet werden kann, innerhalb dessen Objekte rein zufällig algorithmisch bzw. prozedural generiert und mit Eigenschaften ausgestattet werden können, sodass Vorkommen und Eigenschaften gemäß eines definierten Rahmens realistisch erscheinen. Mit der realen Welt und deren Gesetzmäßigkeiten als Vorbild lässt sich dafür ein universelles Modell erstellen, sodass individuelle Spielwelten durch die Angabe von nur wenigen abstrakten Parametern

wie z. B. dem Anteil von Wasser, der Bevölkerungszahl und der Zuordnung von Kontinenten komplett zufällig generiert werden können.



Abbildung 3.2: Platzierung von Palmen

Soll ein Objekt jedoch nicht nur als Dekoration dienen, sodass Widerspruchsfreiheit besteht, sondern eine konkrete Funktion erfüllen, müssen individuellere Eingaben an die Software möglich sein, um die gewünschte Funktionalität zu erreichen, die von Spiel zu Spiel sehr variieren kann. Soll z. B. ein Berg eine bestimmte Stelle verdecken, oder bestimmte Objekte nach spieldaten spezifischen Regeln an bestimmten Orten automatisch platziert werden, so ist eine unkomplizierte Formulierung und konsistente Integration solcher spezieller Anforderungen in ein Modell der Spielwelt, welches allen zur Widerspruchsfreiheit definierten Bedingungen genügt, keine triviale Aufgabe.

Diese Anforderungen an die Eingabemöglichkeiten können nämlich eine oder mehrere Abstraktionsebenen betreffen, in der Präzision sehr variieren und sollten möglichst frei vom Game-Designer bzw. Level-Designer bestimmt werden können, sodass das Modell flexibel sein muss und keiner die Flexibilität beschränkenden Objekthierarchie unterliegen darf.

Als Lösungsansatz zur Architektur wird die Kombination und Weiterentwicklung von Entwurfsmustern verfolgt, da sie zur Lösung von vielfältigen Problemen, bei denen mehrere konkurrierende Aspekte berücksichtigt werden müssen, bewährte Konzepte sind. Entwurfsmuster sind geeignet, einem Entwurf auch auf höheren Abstraktionsebenen Flexibilität zu verleihen [20]. Ein möglichst flexibles Modell wird angestrebt, da es anpassbar und erweiterbar sein soll.

3.2 Modell und Entwurf

Die Spielweltgenerierung soll, nachdem nur wenige Parameter wie z. B. Anteil von Wasser, Bergen etc. eingegeben werden, automatisiert ablaufen, indem zunächst die aus Erde und Wasser bestehende Grundfläche erstellt wird und dann in weiteren

Schritten modifiziert, mit Materialien und Objekten bestückt wird, sodass schließlich das 3D-Modell der generierten Spielwelt dargestellt werden kann. Abbildung 3.3 stellt eine Übersicht über die Hauptschritte des Generierungsprozesses dar.

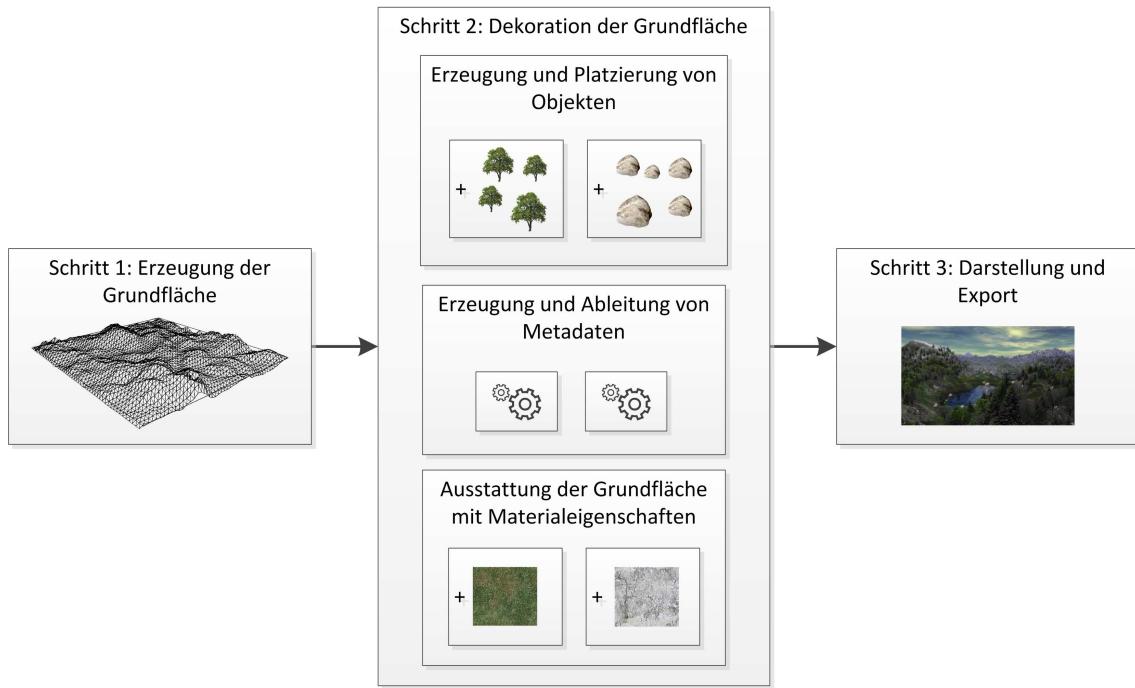


Abbildung 3.3: Schritte des Generierungsprozesses

3.2.1 Analyse

Um ein einheitliches Modell zu entwickeln, welches eine möglichst umfangreiche und komplexe Spielwelt generieren kann, müssen die für den Generierungsalgorithmus relevanten Eigenschaften von Objekten identifiziert und eine geeignete Repräsentation gefunden werden. Von zentraler Bedeutung ist dabei:

- Trotz der Verschiedenheit der Objekte und ihrer möglichen Darstellungsformen durch Abstraktion eine höhere Kohärenz von Daten und Funktionen zu erreichen, sodass diese gekapselt werden können.
- Die Komplexität der Realität auf eine virtuelle Realität zu reduzieren: Die Ästhetik des Spiels bzw. die Spielmechanik und die Optik sowie die technischen Möglichkeiten bilden den Rahmen.
- Erweiterbarkeit und Flexibilität gewährleisten, damit eine Einsetzbarkeit für beliebige Spiele und beliebige Szenen gegeben ist.

Hierfür werden zunächst beispielhaft anhand eines konkreten Objekts und dann generell die Schritte und Einflussfaktoren beim Generierungsprozess untersucht.

Ein Baum wächst in der Natur dort, wo geeignete Klima- und Bodenbedingungen für seine Art herrschen. Da er an einer Stelle wächst, wo ein Samenkorn hingefallen ist, hat auf die genaue Position auch der Zufall Einfluss und im Umfeld werden sich bei geeigneten Bedingungen weitere Bäume derselben Art befinden. Aufgrund des benötigten Lebensraums wachsen in einem Wald Bäume meist mit einem gewissen Abstand zu anderen Bäumen. Werden diese Beschränkungen mit einem quantitativen Wert kombiniert, der angibt, in welcher Dichte eine Baumart an bestimmten Stellen bzw. Gegenden vorkommen soll, lässt sich der Boden automatisch mit Bäumen bedecken.

Für Pflanzen lassen sich die Faktoren, die die Bedingungen beeinflussen, teilweise aus anderen Faktoren ableiten, z. B. herrscht in der Nähe von Gewässern ein feuchteres Klima als in der Wüste. Diese Faktoren lassen sich auf verschiedenen Ebenen erfassen:

1. Die Erdoberfläche (engl. *Terrain*), Art des Kontinents und die Eigenschaft, ob Land oder Wasser
2. Klimabedingungen wie Temperatur, Feuchtigkeit, Wind und Bodenbedingungen
3. Flora und Vegetation: z. B. Bergland, Wald, Wüste, Strand etc. und bei bewohnten Gebieten die Art und der Grad der Bebauung wie z. B. Wohngegend oder Gewerbegegend
4. Bodenmaterial (z. B. Fels, Sand oder fruchtbare Erde bzw. in einer Stadt Baufläche, Straßenfläche etc.)

Im Folgenden werden Modell und Algorithmus zur automatischen Generierung dieser Daten beschrieben.

3.2.2 Räumliches Metamodell

Bei der automatischen Platzierung neuer Objekte in der Spielwelt muss geprüft werden, ob der Ort auch für die Platzierung geeignet ist und das Objekt nicht mit anderen, bereits platzierten Objekten kollidiert. Oder es muss geprüft werden, ob eine entsprechende Docking-Stelle vorhanden ist. Diese Beschränkungen lassen sich als Interaktion der jeweiligen Objekte konkret und strikt modellieren: Die Drahtgittermodelle von zwei Objekten können, wenn sie sich an zwei verschiedenen Punkten im 3D-Raum befinden, entweder kollidieren oder nicht und Straßenschilder sollten z. B. nur an bestimmte Docking-Stellen, die sich an geeigneten Stellen von Straßen befinden, gesetzt werden.

Im Gegensatz dazu ist bei der Modellierung der auf den vier Ebenen erfassten Daten wie Klima, Kontinent etc. aufgrund ihrer abstrakteren Natur und von Zufallseinflüssen nur ein bestimmter Grad an Exaktheit und Striktheit notwendig und sinnvoll.

Unrealistische Konstellationen müssen strikt ausgeschlossen werden, mögliche Konstellationen jedoch sowohl durch Zufallsalgorithmen als auch durch eine Definitionseingabe über die Bedienschnittstelle einfach und flexibel erzeugbar sein. Eine gewisse Unschärfe bzw. Abstraktheit wird dabei in Kauf genommen, da komplexe biologische Gesetze nur äußerst grob visualisiert werden sollen. Um beispielsweise eine Szene zu erzeugen, bei der am Strand Palmen stehen, sind die Anforderungen für das Gameplay und die Optik von Belang. Hier muss das Modell auch mehr oder weniger abstrakte manuelle Modifikationen der Form „an bestimmten Stellen sollen einige Palmengruppen stehen“ zulassen. Diese müssen geeignet formuliert werden können, und vor allem ohne dass Widersprüche zu anderen Modifikationen entstehen, ins Modell eingefügt werden können. Im konkreten Fall bedeutet dies, dass die Erzeugung des Terrains zwar einer der ersten und grundlegenden Schritte ist und einen großen Einfluss auf die meisten anderen Faktoren hat, die Definitionen von manuellen Modifikationen aber unabhängig von konkreten Ausprägungen getroffen werden sollten, damit sie für andere Terrainausprägungen weitestgehend wiederverwendet werden können.

Ziel ist es daher, die Abhängigkeiten der Daten und Bedingungen, für die schon grob vier Ebenen gebildet wurden, zu erfassen und bei Zugeständnis einer gewissen Unschärfe und kleinerer Widersprüche nach Abhängigkeiten, die für Berechnungen benutzt werden können, zu sortieren, d. h. möglichst den Abhängigkeitsgraphen der Einflussfaktoren in eine topologische Sortierung zu überführen. Dadurch kann ein einfaches Modell realisiert werden, anhand dessen die Spielwelt schrittweise automatisiert generiert werden und auf allen Ebenen manuelle Veränderungen und Zufallseinflüsse annehmen kann, sodass nach Änderungen einer beliebigen Ebene komplett automatisiert eine neue Spielweltausprägung berechnet werden kann, ohne dass definierte Konsistenzbedingungen verletzt werden. Abbildung 3.4 zeigt die auf vier Hauptebenen aufgeteilten Zwischengrößen.

Die Datengrößen sind mehr oder weniger abstrakte Zwischengrößen, die für eine funktionale Berechnung von Existenzbedingungen und Ausprägungsfaktoren von Objekten definiert werden. Eine Ebene kann dabei auf die Daten von höhergelegenen Ebenen zurückgreifen, oder auf Daten derselben Ebene, sofern der Abhängigkeitsgraph kreisfrei bleibt.

Dies entspricht insofern nicht der Realität, als komplexe Prozesse mit einer Vielzahl von Faktoren und vor allem zirkulären und wechselseitigen Einflüssen die Entstehung bestimmter Landschaften bzw. Existenzbedingungen und Ausprägungsfaktoren von Objekten bestimmen. Umfassendere und komplexere Modellierungskonzepte sind zahlreich unter den Stichworten *Spatial Analysis* und *Kartographische Modellierung* behandelt worden und stellen eigene Themenfelder dar (vgl. [44], [16]).

Um den Grad der Automatisierung bei der Erzeugung zu maximieren, ist ein entsprechendes unidirektionales Teilmmodell für die Abhängigkeiten sinnvoll, denn so kann der Kernprozess logisch bzw. mathematisch modelliert und implementiert werden. Abweichungen können an allen Stellen durch manuelle Modifikation oder zusätzli-

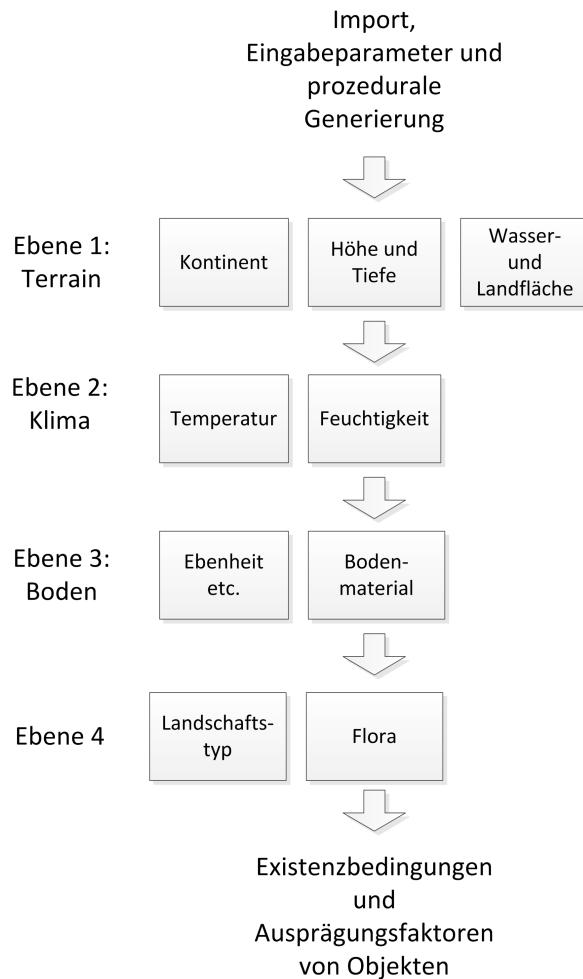


Abbildung 3.4: Metadatengenerierung

che Modelle eingefügt werden. Komplexere Modelle, die quantitativ und qualitativ exakter arbeiten, hätten an dieser Stelle ein höheres Risiko zur Folge, die für die Architektur bzw. den Softwareentwurf wesentlichen Aspekte aus den Augen zu verlieren.

Die Gesamtheit der generierten Daten, die in dem Modell aus den Eingabeparametern erzeugt werden können, bildet einen mehrdimensionalen Datenraum aus Metainformationen der zu generierenden Spielwelt, aus denen sich für jeden Ort Existenzbedingungen und Ausprägungsfaktoren von Objekten und anderen Phänomenen der Spielwelt ableiten lassen. Abbildung 3.5 zeigt ein Beispiel des Metamodells mit vier Datenebenen.

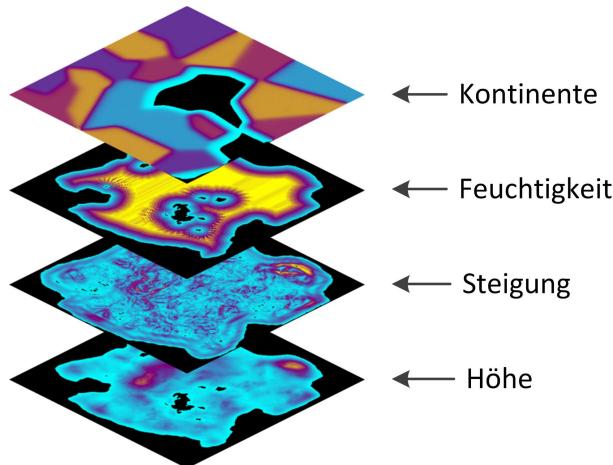


Abbildung 3.5: Räumliches Metamodell mit vier Ebenen

3.2.3 Architektur und Komponenten

In diesem Abschnitt wird auf Grundlage des Modells eine Software-Architektur entworfen, die Einteilung in Hauptkomponenten vorgenommen und der Einsatz von Techniken, Mustern und Algorithmen erläutert.

Architektur

Die Architektur soll die Komplexität domänengerecht reduzieren, aber auch in Anwendung und Konfiguration einfach und effizient zu bedienen sein, dafür werden drei Gruppen von Anwendungs- bzw. Modifikationsfällen unterschieden, diese sind in Tabelle 3.1 dargestellt. Aufgrund von Abstraktionslevel und Häufigkeit berücksichtigt die Architektur das *Open-Closed Prinzip* ([30], S. 57) nur für Änderungen an der statischen Struktur konsequent: Alle abstrakten Daten und auch deren Semantik sollen nicht nur erweiterbar, sondern auch änderbar sein.

Gruppe	Anwendungsfall	Abstraktionslevel	Häufigkeit
1	Eingabe einer abstrakten Weltdefinition	Abstrakt	Sehr oft
2	Konfiguration (Ändern, Hinzufügen, Löschen) von z. B. Metadimensionen, Materialien, Objektgruppen	Abstrakt	Oft
3	Änderungen an der statischen Struktur: Klassen, Interfaces und Quelltext	Konkret	Selten

Tabelle 3.1: Anwendungs- und Modifikationsfälle

In Abbildung 3.6 ist eine Übersicht über die Architektur und den Hauptdatenstrom dargestellt. Im ersten Schritt wird die Grundfläche der Spielwelt generiert

oder eingelesen. Dann werden daraus die Metadaten erzeugt. Der Controller arbeitet dann nacheinander schrittweise die Befehle ab und modifiziert je Befehlsschritt die Spielwelt-Daten. Er greift dazu auf die Metadaten und die persistenten Daten, wie z. B. 3D-Models und Materialien zurück, um Objekte in die Spielwelt einzufügen. In den folgenden Abschnitten wird die Funktionsweise der einzelnen Komponenten erläutert.

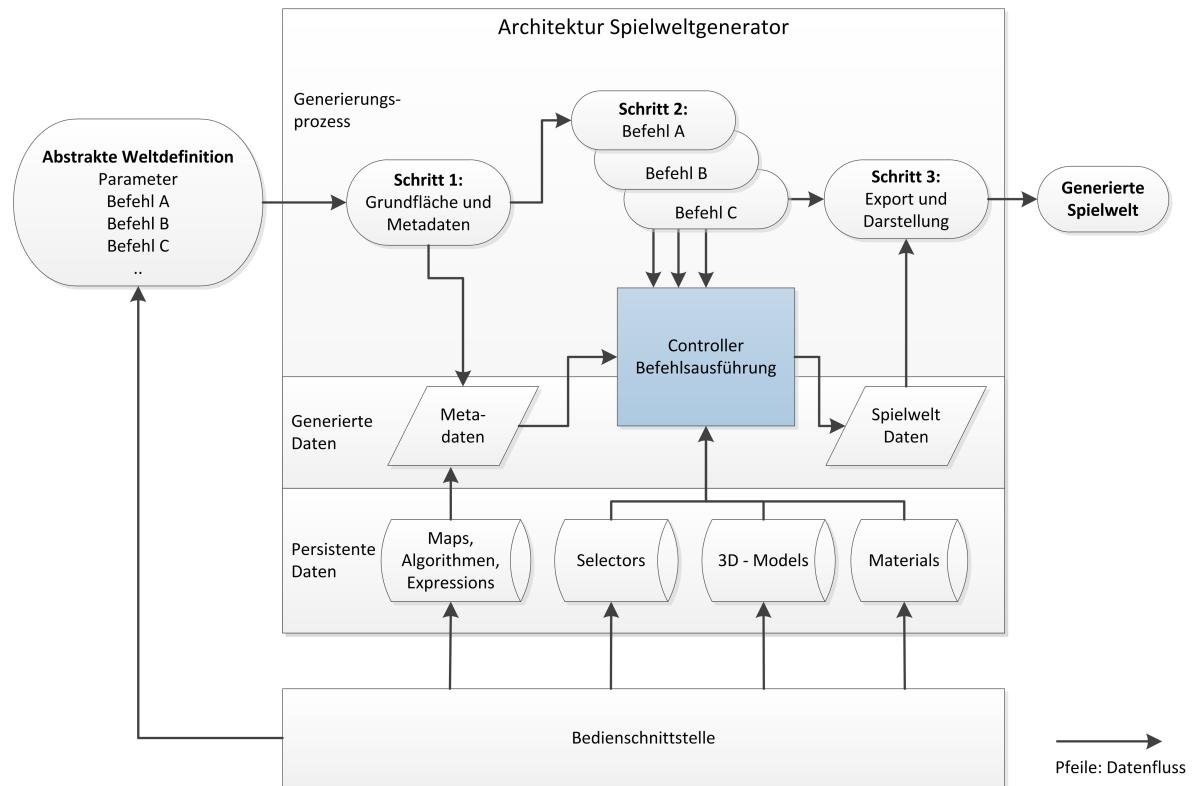


Abbildung 3.6: Architekturübersicht

Abstrakte Weltdefinition

Die abstrakte Weltdefinition ist der Eingabeparameter des Generierungsprozesses und beinhaltet Befehle, die der Reihe nach ausgeführt werden. Die Befehle beinhalten bzw. referenzieren Ausdrücke (engl. *Expressions*), Algorithmen, 3D-Modelle (engl. *3D-Models*), Materialien (engl. *Materials*) und andere persistent gespeicherte Daten. Eine solche Weltdefinition ist in Abbildung 3.7 beispielhaft, umgangssprachlich formuliert dargestellt und illustriert.

Flexibilität der Spielwelt

Die Flexibilität der Spielwelt wird durch die Definitionssprache realisiert, die im nächsten Abschnitt formal definiert wird. Sie dient der formalen Beschreibung von

folgenden Daten:

- *Existenzbedingungen und Ausprägungsfaktoren*, die Objekte in der Spielwelt an bestimmten Orten annehmen können. Diese Ausdrücke sind in den Selector-Objekten gekapselt, beziehen sich auf die Metadaten und berechnen im Objektgenerierungsprozess die für bestimmte Objekte gültigen räumlichen Bereiche in der Spielwelt und stattet neue generierte Objekte mit bestimmten Eigenschaften aus, die aus den Metadaten berechnet werden.
- *Definition von Metadaten-Dimensionen*: Jede Dimension bzw. Ebene der Metadaten besteht aus einem Ausdruck, der eine Regel angibt, nach der die Daten der Dimension aus der Grundfläche, zufällig generierten Daten und gespeicherten Daten erzeugt werden. Da die Anzahl der Dimensionen nicht beschränkt ist, lassen sich zusätzliche Dimensionen einfügen und das Metamodell beliebig erweitern, sofern keine zirkulären Abhängigkeiten erzeugt werden.

Definitionssprache

Zur abstrakten Definition der Spielwelt wird eine sehr einfache, deklarative *domänen spezifische Sprache* (engl. *Domain Specific Language*) entworfen (vgl. [16]). Dadurch wird die nötige Flexibilität und trotzdem eine relativ einfache Bedienbarkeit erreicht. Die Sprache muss für die beiden Zwecke geeignet sein, zum einen für Ausdrücke, die auf den Metadaten Bereiche der Spielwelt definieren (*Selektierung* von räumlichen Bereichen), und zum anderen Ableitungs- bzw. Berechnungsregeln zur Berechnung von Metadaten aus anderen Metadaten, gespeicherten Datenfeldern oder Zufallsalgorithmen beschreiben können (*Generierung* von Metadaten).

Die Syntax der Sprache ist hier als kontextfreie Grammatik definiert:

```

<ausdruck>      : : = <wert> | <variable> |
                  <klammerung> | <funktion>
<klammerung>    : : = '(' <ausdruck> ')'
<wert>          : : = [0-9]+ | [0-9]+ ',' [0-9] +
<variable>       : : = [A-Z] +
<funktion>       : : = <operator> '[' <argumente> ']',
<argumente>      : : = <ausdruck> | <argumente> ';' <ausdruck>
<operator>        : : = '+' | '*' | '-' | '>' | '<' |
                     'AND' | 'OR' | '=' |
                     <individueller_operator>
<individueller_operator> : : = [A-Z] +

```

Da die Operatorenliste durch individuelle Operatoren und Funktionen erweiterbar sein soll, die mehr Funktionalität kapseln als einfache Rechenoperationen, wird die Syntax von individuellen Funktionen und Variablen nicht genauer präzisiert.

Eine Variable stellt den Identifikator eines gespeicherten 2D-Datenfelds oder einer Ebene des Metamodells dar, welches ebenfalls 2D-Daten liefert. Die beiden Anwendungszwecke für Ausdrücke der Sprache, Auswahl und Generierung von Metadaten, werden im Folgenden beschrieben.

Auswahl von Metadaten: Der Ausdruck wird in einer geschachtelten Iteration für alle (x, y) -Werte der im Ausdruck beschriebenen 2D-Datenfelder ausgewertet, sodass ein Boolesches 2D-Datenfeld generiert wird, das eine Selektierung eines Bereichs auf der Grundebene der Spielwelt darstellt: *True* bzw. 1 bedeutet, dass die Position (x, y) Teil der Selektion ist und *False*, dass die Position (x, y) nicht zur Selektion gehört.

Generierung von Metadaten: Ein solcher Ausdruck wird analog ausgewertet, nur werden bei der Auswertung keine Booleschen Werte, sondern Zahlenwerte erzeugt.

Die Operatoren sind allgemein für Kommazahlen, ganze Zahlen und Boolesche Werte definiert, die auch gemischt werden können, indem sie vom Interpreter automatisch konvertiert werden. Je nachdem, welches Format der Ausdruck bzw. Operator verlangt, werden Boolesche Werte automatisch in Zahlenwerte umgewandelt, oder umgekehrt, d. h. *TRUE* \Leftrightarrow 1.0 und *FALSE* \Leftrightarrow 0.0³.

Weitere formale und semantische Details zur Definitionssprache und den einzelnen Operatoren werden im Teil der Implementierung behandelt.

Die hier sehr allgemein gefasste Sprachdefinition kann als unpräzise erscheinen, weil z. B. alle Operatoren bzw. Funktionen zusammengefasst sind und die Anzahl der Argumente nicht in der Sprache festgelegt ist, obwohl die einzelnen Operatoren unär oder binär sind oder eine bestimmte Anzahl von Argumenten verlangen.

Ziel der Verallgemeinerung ist, die Sprache so einfach und intuitiv wie möglich zu gestalten: Damit ohne Detailkenntnisse kompakte, aber aussagekräftige Ausdrücke erstellt werden können, ist es von Vorteil, wenn die Sprache die Präzision und Komplexität der Schnittstellen und Daten, auf denen sie definiert ist, widerspiegelt: Ein- und Ausgabedaten des Modells sind abstrakter Natur und auf dieser Abstraktionsebene nicht typisiert. Die Anwendung erfolgt experimentell, auch sind die Operatoren bzw. Funktionen erweiterbar. Eine höherer Grad an Konkretisierung und damit Formalisierung würde zu einer unnötigen Komplexität sowohl der Sprache als auch der Implementierung des Interpreters und der Datenobjekte führen.

Abbildung 3.8 zeigt ein Beispiel, bei dem durch Anwendung des Gleichheits-Operators auf Boolean-Werte des Datenfelds `WATER` und Zahlenwerte des Datenfelds `CONTINENT` ein neues Datenfeld berechnet wird.

Persistente Daten

Die auf dieser Schicht befindlichen Komponenten haben die Funktion, Daten und Medien für die Operationen, die vom Controller ausgeführt werden, zur Verfügung

³Zahlenwerte ungleich 0 und 1 werden zu *FALSE* konvertiert

zu stellen und werden in Tabelle 3.2 erläutert. Die Daten können über die Bedienschnittstelle erstellt, modifiziert und importiert werden. In Abbildung 3.9 sind Visualisierungen der Daten, die von den unterschiedlichen Komponenten verwaltet werden, dargestellt.

Komponente	Funktion
Maps	Speichert 2D-Datenobjekte von erzeugten oder importierten Daten, z. B. von Heightmaps ⁴ persistent ab, so in Ausdrücken auf diese zurückgegriffen werden kann.
Algorithmen	Speichert Algorithmenobjekte mit Parametern, die zur Generierung oder Modifikation von Daten, wie z. B. Heightmaps verwendet werden.
Expressions	Speichert Ausdrücke (in der Definitionssprache) zur Definition und Erzeugung von Metadaten.
Selectors	Speichert Ausdrücke (in der Definitionssprache) zur Selektierung von Bereichen der Spielwelt-Daten, damit Objekte nur oder vorwiegend in bestimmten Bereichen platziert werden können.
3D-Models	Speichert 3D-Daten, Texturen und Eigenschaften von Objektklassen, die zur Erstellung von Objekten in der Spielwelt benötigt werden.
Materials	Speichert Materialien mit Texturen, Farb- und Lichtreflexionseigenschaften.

Tabelle 3.2: Komponenten zur Verwaltung von Daten und Medien

Erzeugung und Platzierung von Objekten

Die zentrale Aufgabe des Controllers besteht in der Generierung von Objekten bzw. Objektmengen und Materialflächen in der Spielwelt.

Ein Befehl zum Generieren von Objektmengen beinhaltet folgende Daten:

- Ausdrücke zur Selektierung bestimmter Bereiche, in welche die Objekte gesetzt werden sollen
- 3D-Modelle der Objekte, die generiert werden sollen
- Anzahl der zu generierenden Objekte oder relativer Bedeckungsanteil
- zusätzliche Parameter, die Eigenschaften der Objekte bestimmen bzw. modifizieren, z. B. ob die Objekte zufällig oder regelmäßig angeordnet werden sollen

Ein Befehl zum Generieren einer Materialfläche ist analog aufgebaut und enthält statt 3D-Modellen die Materialeigenschaften bzw. eine Referenz auf ein Materialobjekt.

Die Ausführung eines Befehls läuft in Form von einer Iteration ab, die an geeigneten Positionen so viele Objekte generiert, bis die definierte Anzahl von Objekten oder

der definierte Bedeckungsanteil erreicht ist. Wenn keine geeigneten Stellen mehr vorhanden sind, bricht die Iteration ab.

Die geeigneten Positionen werden in den folgenden Schritten bestimmt:

1. Erst wird anhand des Selectors der im aktuellen Zustand der Spielwelt für das Objekt zulässige Bereich bestimmt. Dazu werden mehrere Ausdrücke des Selectors, die sich auf die Metadaten oder auf Daten der Spielwelt⁵ beziehen können, ausgewertet. Anhand von mehreren Ausdrücken lassen sich die Bereiche mehrstufig bestimmen, sodass nicht nur die Bedingung, ob eine Position zulässig ist oder nicht, bestimmt werden kann, sondern in den zulässigen Bereichen mehrere Stufen erzeugt werden können, die Bereiche unterscheiden, in denen Objekte in einer dichteren Häufung als in anderen Bereichen oder mit bestimmten Eigenschaften (z. B. in bestimmten Größen oder mit bestimmten Dekorationen) generiert werden sollen (vgl. Abbildung 3.9, Selectors).
2. Im zulässigen Bereich wird z. B. zufällig eine Position ausgewählt und weitere Prüfungen, ob an dieser Position ein entsprechendes Objekt generiert werden kann, durchgeführt:
 - Kollision: Es wird geprüft, ob an der Stelle bereits andere Objekte vorhanden sind, mit denen das neue Objekt kollidieren würde.
 - Docking: Bei Objekten, die an andere gekoppelt werden müssen, wird im Umfeld nach gültigen Stellen bzw. Bereichen gesucht und ggf. die Position entsprechend modifiziert. Landschaftsobjekte wie z. B. Bäume werden auf die Grundfläche gesetzt, Objekte, die auf dem Wasser schwimmen, auf die Wasseroberfläche etc.

Wenn die Zulässigkeit erfolgreich geprüft wurde, wird eine Objektinstanz eines der vom Befehl bestimmten 3D-Modelle erstellt und an der gewählten Position in die Spielwelt-Daten eingefügt. Damit bei einem Wald beispielsweise nicht alle Bäume gleich aussehen, ist es möglich, dass die Objektinstanz nicht immer mit demselben 3D-Modell, sondern z. B. zufällig oder abwechselnd mit einem aus einer Gruppe von 3D-Modellen ausgewählten 3D-Modell erstellt wird. Zusätzlich wird die Objektinstanz mit bestimmten Eigenschaften wie Größe, Farbe etc. dekoriert, die entweder zufällig bestimmt werden oder sich nach bestimmten Daten im Umfeld der Position richten, sodass z. B. bestimmte Baumarten in kälteren Regionen weniger Blätter tragen und eine geringere Höhe erreichen.

In Abbildung 3.10 sind diese Schritte für Befehl C aus Abbildung 3.7 (*Füge bei „einigen halbwegs ebenen Flächen, die sich auf mittlerer Höhe befinden“, Bäume hinzu*), demonstriert.

⁵nicht implementiert, vgl. 4.3.2

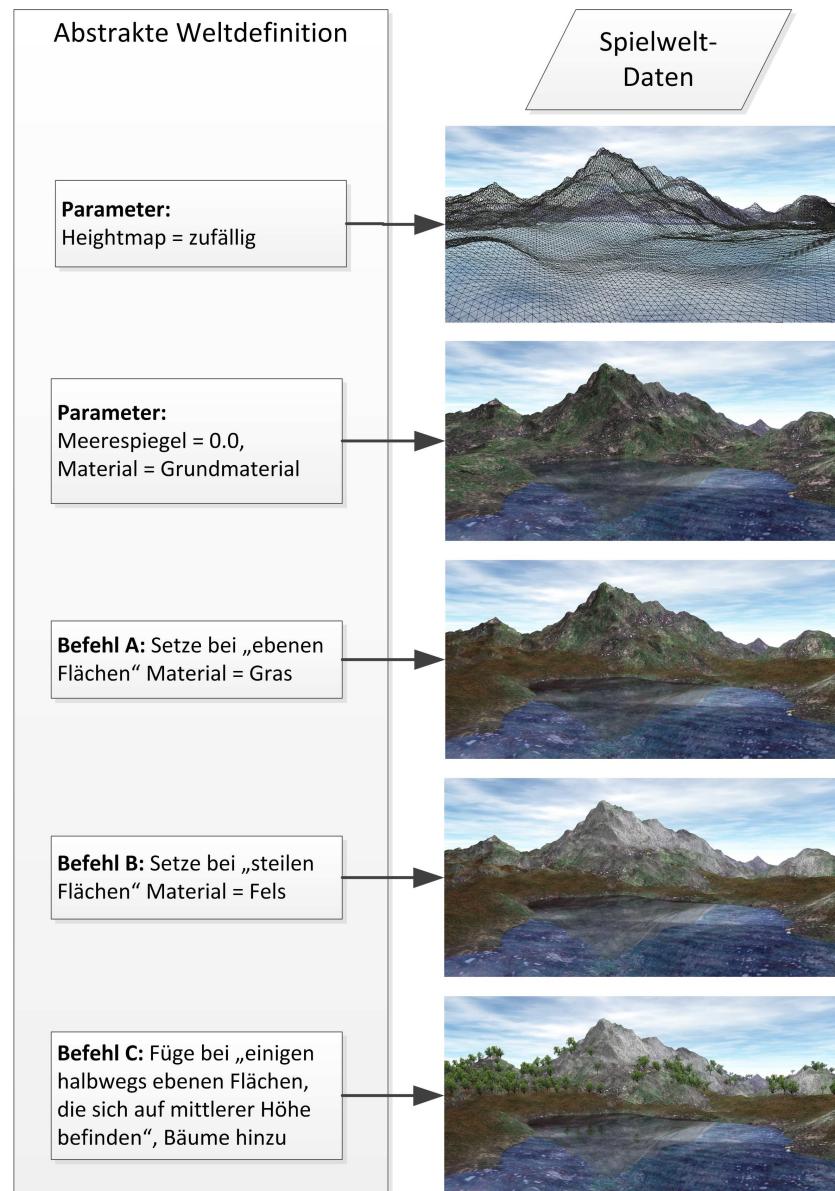


Abbildung 3.7: Abstrakte Weltdefinition und erzeugte Spielwelt-Daten

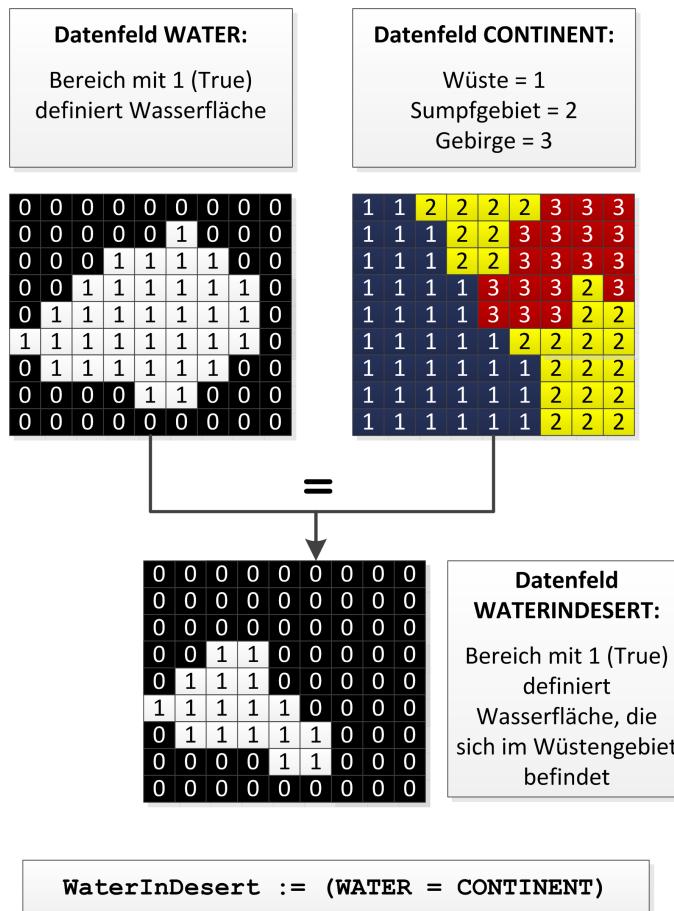


Abbildung 3.8: Definition und Berechnung der neuen Ebene WaterInDesert

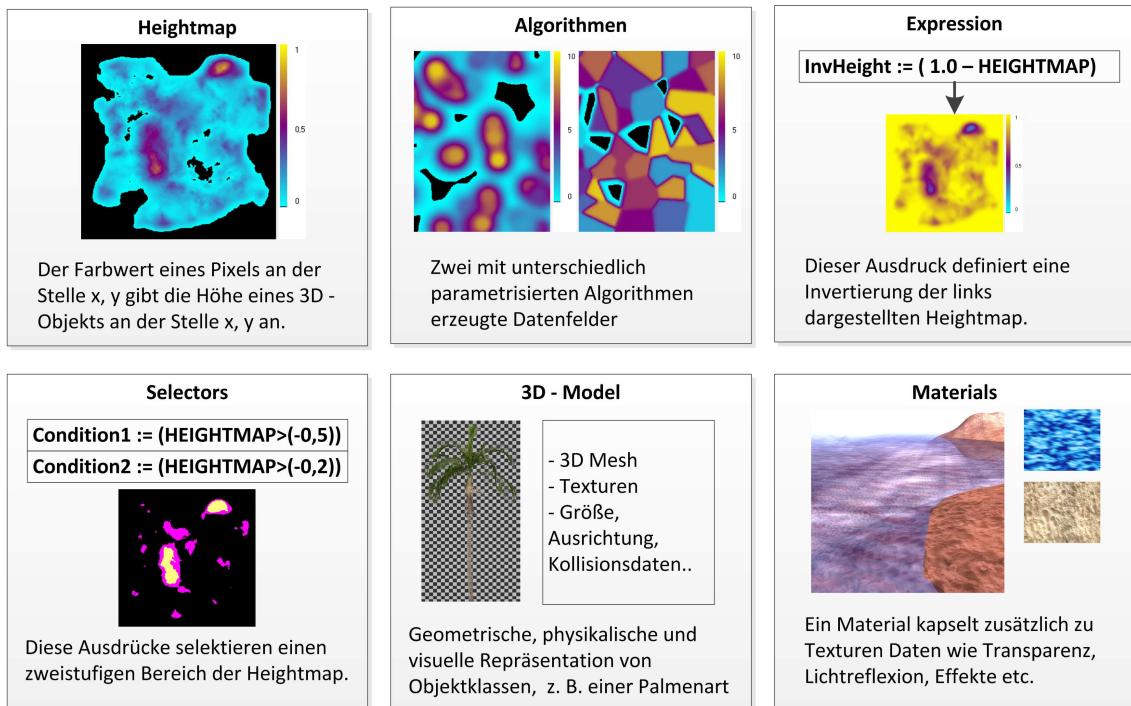


Abbildung 3.9: Visualisierungen der verwalteten Daten

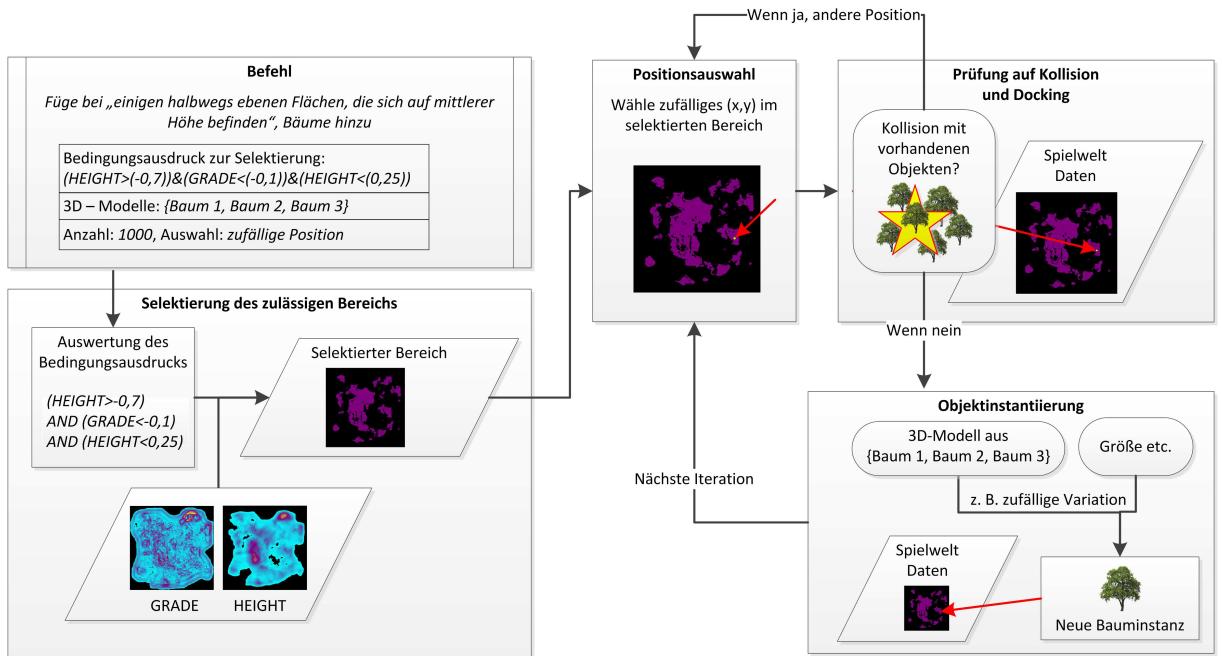


Abbildung 3.10: Schritte der Objektgenerierung und -platzierung

3.3 Implementierung

In diesem Abschnitt werden Details der Implementierung des beschriebenen Modells beschrieben, soweit sie sich auf Besonderheiten des Modells und der Architektur beziehen oder im Zusammenhang von Softwaretechnik und Spieleentwicklung relevant sind. Funktionalität, die den Aufgabenbereich von komplexen Grafik-, Physik- und Spieleengines wie Grafikdetail, Effizienz, Kompatibilität, Plattformen oder Objekt-design betrifft, werden in dieser Arbeit nicht behandelt⁶.

3.3.1 Metadaten

Die Daten des Metamodells werden zum Zweck einer einfachen Visualisierung und Verarbeitung je Dimension in einem 2D-Datenfeld repräsentiert. Sie können so auch als Grafik importiert, exportiert und mit anderer Software, z. B. mit Bildverarbeitungsoperatoren modifiziert werden.

Metadaten-Klasse

Die Metadaten-Klasse beinhaltet die Liste der einzelnen Ebenen. Jede Ebene besitzt einen Namen, der als Identifikator dient und einen Ausdruck, der die Herleitung der jeweiligen Ebene bestimmt. Jede Ebene speichert intern den zulässigen Datenbereich. Über die Schnittstelle der Metadaten-Klasse können neue Ebenen hinzugefügt, die Definitionen geändert, und einzelne oder die Gesamtheit der Ebenen neu berechnet werden. Abbildung 3.11 zeigt die Eingabemaske, über die die Einstellungen festgelegt werden können.

Definition der Metaebenen

In Tabelle 3.3 sind die Definitionen der Metaebenen aufgelistet, die zur Erstellung einiger Landschaften verwendet wurden.

⁶vgl. 3.1.4 Abgrenzung und Kompromisse

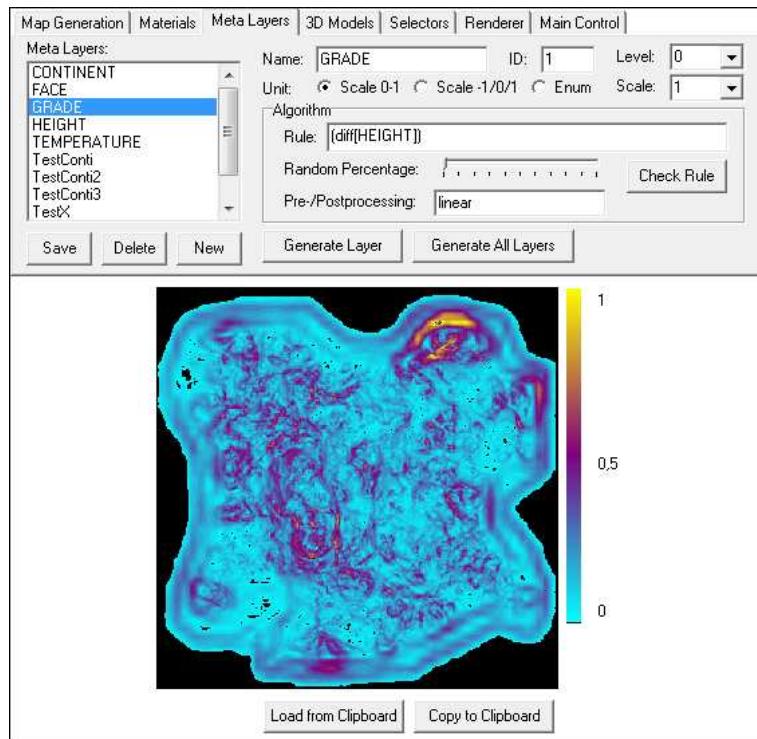


Abbildung 3.11: Bedienschnittstelle zur Metadaten Definition

Metaebene Identifikator	Bedeutung	Ausdruck zur Herleitung
HEIGHT	Höhe bzw. Tiefe der Grundfläche	HEIGHT \leftarrow random[...]
CONTINENT	Kontinent: Wüste, Gebirge, Flachland, Pol, Tropen	CONTINENT \leftarrow random[...]
GRADE	Steigung der Grundfläche	GRADE \leftarrow difference[HEIGHT]
SURFACE	Oberfläche: Werte unter 0 bedeuten Wasser, Werte über 0 bedeuten Land	SURFACE \leftarrow (distance[HEIGHT;-1;0])*0,1
TEMPERATURE	Temperatur in Grad Celsius, abhängig vom Kontinent	TEMPERATURE \leftarrow transform[CONTINENT;0;50;1;20;...]
WET	Feuchtigkeit, abhängig von Kontinent und Nähe zum Wasser	WET \leftarrow transform[CONTINENT;0;0;1;0,3;...]* (SURFACE)*0,7+(SURFACE*0,2)-0,1
WIND	Windstärke, abhängig von Steigung, Oberfläche und Kontinent	WIND \leftarrow (transform[CONTINENT;0;0,3;1;0,7..]*0,7*SURFACE)-(GRADE*0,2)+(SURFACE*0,3)

Tabelle 3.3: Ebenen des Metadaten-Modells

Die Ebenen HEIGHT und CONTINENT werden zufällig erzeugt oder können auch importiert bzw. manuell festgelegt werden. Während die Daten einiger Ebenen intuitive physikalische Maßeinheiten besitzen, wird z. B. bei CONTINENT die Kontinentzugehörigkeit durch Zahlenwerte wie Wüste = 0, Gebirge = 1, etc. dargestellt und Ebenen wie WET und WIND bezeichnen nur abstrakt Feuchtigkeit und Windstärke, um eine grobe Unterscheidung zwischen z. B. trockenen und feuchteren Gebieten treffen zu können.

Abbildung 3.12 stellt die Abhängigkeiten der konstruierten Ebenen dar: Die Pfeile zeigen auf die Ebenen, deren Daten zur Berechnung einer anderen Ebene benötigt werden.

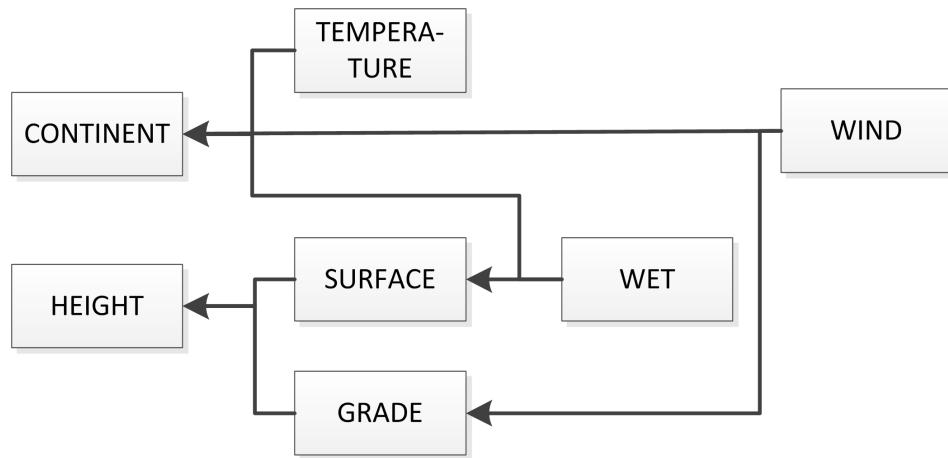


Abbildung 3.12: Abhängigkeiten der Metaebenen

Die in den Ausdrücken enthaltenen Operatoren `difference`, `distance` und `transform` berechnen die Rückgabewerte anhand von Daten der im Parameter angegebenen Ebene, z. B. wird für `GRADE = difference[HEIGHT]` die Steigung aus den Höhen-daten im lokalen Umfeld der jeweiligen (x,y)-Position abgeleitet. Sie fungieren auf den Daten als lokale Operatoren, die speziell parametrisiert werden können. Die implementierten Operatoren werden im nächsten Abschnitt einzeln aufgeführt.

3.3.2 Interpreter

Der Interpreter wurde als direkte Abbildung der für die Definitionssprache entworfenen Grammatik implementiert mit der Erweiterung, dass binäre Operatoren abweichend von der Syntaxdefinition `<operator>[<argument1>;<argument2>]` in der üblichen Form `<argument1><operator><argument2>` dargestellt werden können und für `AND`, `OR` und die individuellen Operatoren Kurzformen verwendet werden können. Die verwendeten binären und individuellen Operatoren sind in den Tabellen 3.4 und 3.5 aufgelistet.

Operator	Kurzform	Bedeutung
$+, *, -$	$+, *, -$	Rechenoperationen auf Daten
AND	&	Logisches AND
OR	\sim	Logisches OR
$<, >, =$	$<, >, =$	Logische Vergleichsoperatoren

Tabelle 3.4: Binäre Operatoren

Operator	Kurzform	Parameterliste und Bedeutung
transform	t	[Datenfeldidentifikator; Wert1; Wert2; ...] Transformiert in der Parameterliste enthaltene Werte aus dem angegebenen Datenfeld, $Wert1 \rightarrow Wert2$, $Wert3 \rightarrow Wert4, \dots$ $result = transform[data; a1; a2; a3; a4]$ $\Rightarrow result(x, y) = \begin{cases} a2 & \text{für } data(x, y) = a1 \\ a4 & \text{für } data(x, y) = a3 \\ \dots & \dots \end{cases}$
random	r	[Zufallsalgorithmusidentifikator] Generiert Zufallsdaten.
difference	d	[Datenfeldidentifikator] Berechnet die lokale Ableitung der Daten durch Filterung mit Sobel-Operator.
get	g	[Datenfeldidentifikator] Liest Daten aus einem gespeicherten Datenfeld.
distance	e	[Datenfeldidentifikator; Wert1; Wert2] Berechnet näherungsweise die minimale Distanz (euklidischer Abstand), in der sich im Datenfeld ein Wert im Bereich zwischen Wert1 und Wert2 befindet: $result(x, y) = \min \left(\sqrt{(x' - x)^2 + (y' - y)^2} \right) \text{ mit } data(x', y') \geq Wert1 \wedge data(x', y') \leq Wert2$

Tabelle 3.5: Individuelle Operatoren und Bedeutung

3.3.3 Renderer

Um über die Bedienschnittstelle ohne Verzögerung das optische Resultat von Änderungen an der Spielwelt zu erhalten, wird diese in Echtzeit gerendert, wobei die

Kameraperspektive frei eingestellt werden kann. In diesem Abschnitt wird die 3D-Darstellung der Spielwelt-Daten erläutert.

Meshs und Materials

Die Daten von Grundfläche und Wasserfläche sind als sog. *Meshs* gespeichert, die in einem 2D-Feld die 3D-Koordinaten aller Knoten eines Objekts enthalten. Die Materialzuordnung jedes Knotens wird in drei Datenfeldern MAT1, MAT2 und MAT3 gespeichert, wobei MAT1 und MAT2 zwei Materialidentifikatoren speichern und den prozentuellen Anteil vom ersten Material gegenüber dem zweiten beinhaltet: Ein Knoten kann somit einen Übergang zwischen zwei Materialien darstellen.

In der Renderprozedur werden die Knoten zeilenweise der Reihe nach mit ihren Materialeigenschaften in Form von *Triangle Strips*, einer Kette aus miteinander verbundenen Dreiecken, gerendert.

Abbildung 3.13 zeigt ein visualisiertes Beispiel der Datenfelder und die gerenderte Grafik.

3D-Models

Zur Vereinfachung werden die 3D-Models als flache, teiltransparente Grafik gerendert, die automatisch in Kamerarichtung gedreht wird und die Kollision wird anhand der Objektgröße und durch zwei unterschiedliche Kollisionsradien, einem weichen und einem harten, angenähert: Die weichen, äußeren Kollisionsradien von zwei Objekten dürfen einander überlappen, falls ein Objekt andernfalls nicht erstellt werden kann. Die inneren Radien von zwei Objekten dürfen nicht überlappen. Abbildung 3.14 zeigt die vereinfachte Repräsentation eines Baumes mit den Kollisionsradien.

3.3.4 Spielwelt-Daten

Die Spielwelt-Daten sind als Liste aus Objekten der Grundfläche, Wasserfläche und der darauf generierten Spielwelt-Objekte aufgebaut. Damit es möglich ist, eine hohe Anzahl von Spielwelt-Objekten zu rendern und effizient zu verarbeiten, sind Spielwelt-Objekte einer bestimmten Art zu einer Objektgruppe zusammengefasst, die nach dem Flyweight-Muster aufgebaut ist: Ein Manager-Objekt verwaltet eine Liste von Datenobjekten, in denen jeweils nur spezifische Daten wie Position, Größe und Model-Identifikator enthalten sind. Dadurch wird zur Laufzeit Speicher gespart und eine schnellere Verarbeitung ermöglicht. Ein Wald, der z. B. aus 5000 Bäumen desselben 3D-Models besteht, muss somit nur solche Daten 5000-mal in den Speicher laden, bei denen sich die einzelnen Bäume unterscheiden.

Abbildung 3.15 zeigt ein vereinfachtes Klassendiagramm: Ein Objekt der Klasse FlyweightCloud beinhaltet FlyweightModifier-Objekte. Diese können sehr zahlreich sein und kapseln daher nur die nötigsten Daten.

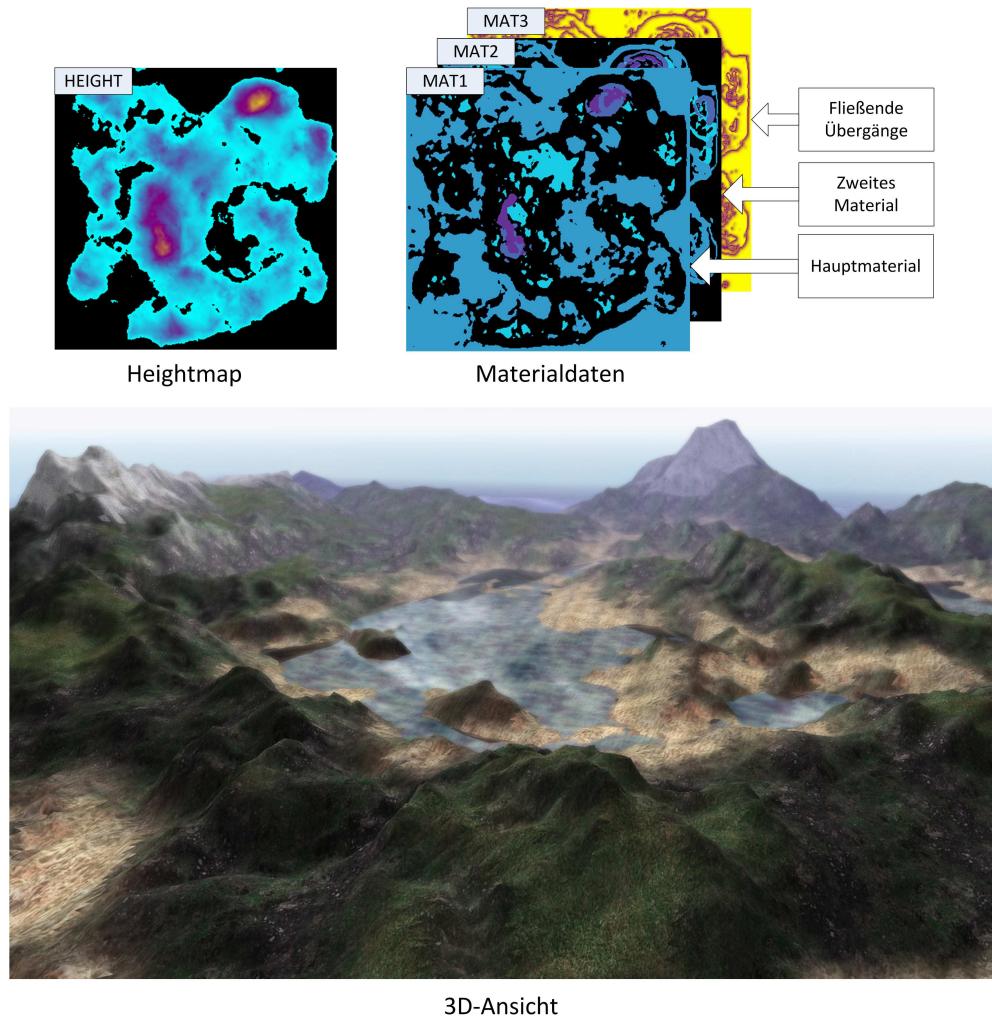


Abbildung 3.13: Materialdaten

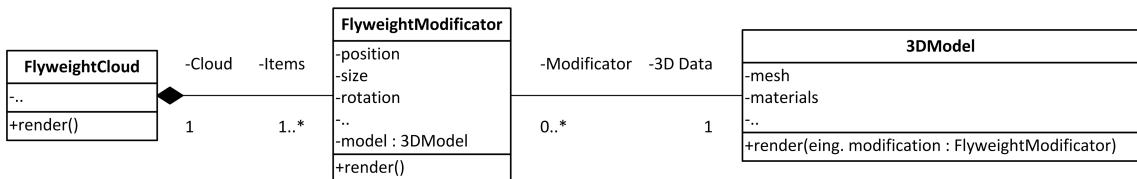


Abbildung 3.15: Statische Struktur der Flyweight-Objekte

Der Renderaufruf ist wegen der potentiell hohen Anzahl von Landschaftsobjekten besonders zeitkritisch, in Abbildung 3.16 ist dieser dargestellt: Die FlyweightModifierator-Klasse kapselt nicht nur möglichst wenig Daten, sondern auch minimale Funktionalität. Die Modifikationsdaten werden beim Renderaufruf dem 3D-Modell übergeben, über welches die eigentliche Renderprozedur erfolgt.

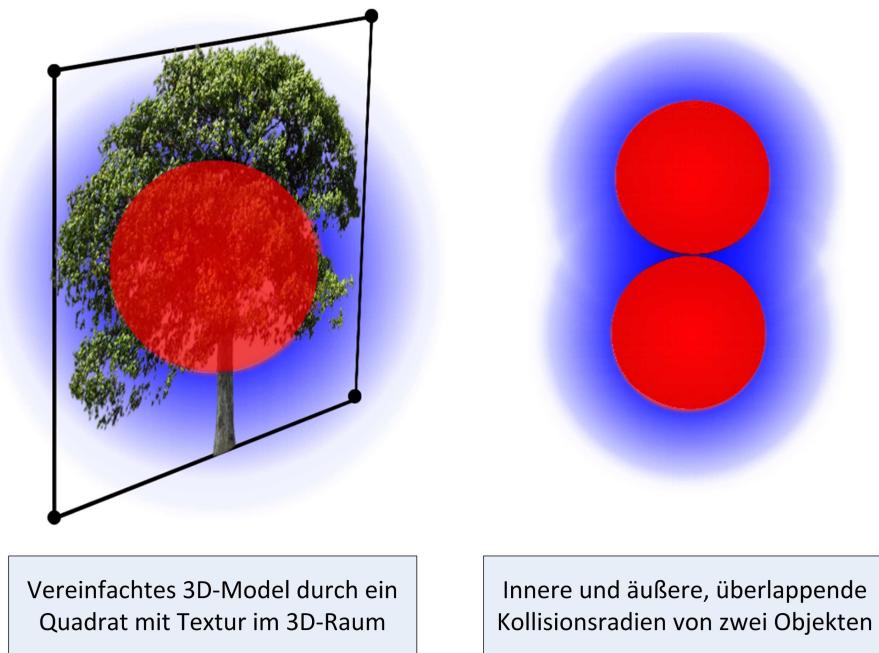


Abbildung 3.14: 3D-Models Vereinfachung und Kollisionsradien

3.3.5 Bedienschnittstelle

In diesem Abschnitt wird ein Beispiel der Definition und Generierung einer Spielwelt über die Bedienschnittstelle beschrieben. Im Anhang sind weitere Beispiele und Funktionen der Bedienschnittstelle aufgeführt. Als Spielwelt wird in diesem Abschnitt eine Berglandschaft mit Bäumen und kleineren Seen erstellt.

Definition von Selectors und Spielwelt

Ein Selector wird über die Bedienschnittstelle durch einen oder mehrere Selektionsausdrücke, durch die Auswahl und Anzahl von 3D-Modellen sowie die Auswahl von einem Material definiert. Als Spielwelt soll eine Landschaft erstellt werden, in die Laub-, Nadelwälder, Bodenpflanzen und Steine gesetzt werden, und in der Berghänge mit Felsmaterial und Berggipfel mit Schnee ausgestattet werden und das Bodenmaterial mit Material, die dem Bewuchs entsprechen, gestaltet wird (also Wald mit Waldboden etc.). In Tabelle 3.6 sind die dazu definierten Selectors aufgeführt. Sie definieren eine Spielwelt als Liste von Befehlen, die jeweils Selector, eine Aggregation von 3D-Models, Anzahl und Material enthalten.

Über die Bedienschnittstelle kann eine Vorschau des selektierten Bereichs und der Kollisionsfläche der zu generierenden Objekte angezeigt werden, Abbildung 3.17 zeigt dies beispielhaft für den Selector *Berge-Nadelwald*: Links der markierte Bereich und rechts eine Vorschau mit den Kollisionsdaten der zu platzierenden Objekte.

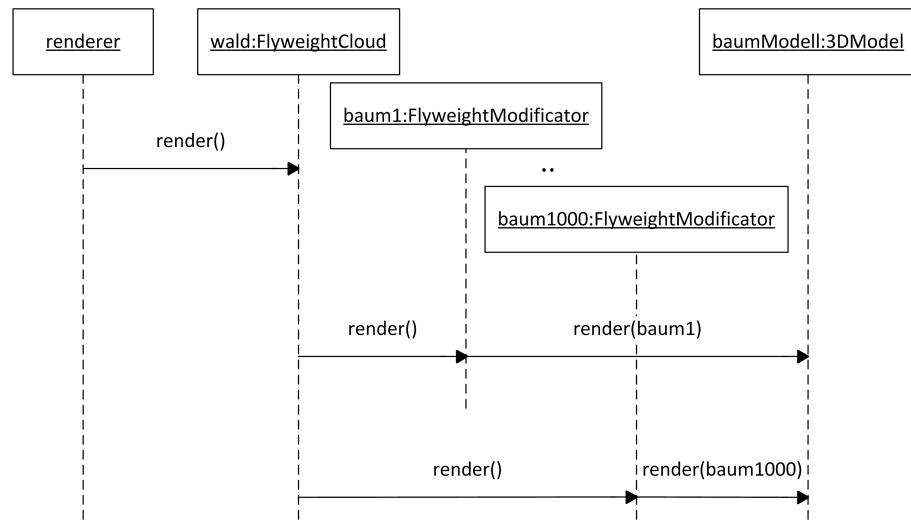


Abbildung 3.16: Flyweight-Objekte: Renderaufruf

Automatischer Generierungsprozess

Anhand der Definitionen aus Tabelle 3.3 wird nun eine Heightmap zufällig generiert, dann werden die Metadaten berechnet und schließlich die Objekte generiert. Abbildung 3.18 zeigt ein gerendertes Bild der erstellten Spielwelt.



Abbildung 3.18: Generierte Landschaft

Selector	Selektierungsausdruck	3D-Models	Anzahl Objekte	Material
Berge-Schnee	(GRADE<-0,3) AND (HEIGHT>-0,2)	-	-	Schnee
Berge-Fels	(GRADE>0,05) AND (GRADE<0,7)	-	-	Fels
Berge-Nadelwald	(HEIGHT>0,9) AND (GRADE<0,0) AND (HEIGHT<0,25)	Tanne1, Tanne2	2000	Waldboden
Berge-Laubwald	(HEIGHT>-0,9) AND (HEIGHT<-0,3) AND (GRADE<-0,6) AND (TEMPERATURE>0,5)	Baum1, Baum2, Baum3, Baum4, Baum5	2000	Laub
Berge- Bodenpflanzen	(HEIGHT>-0,97) AND (HEIGHT<-0,3) AND (GRADE<-0,6)	Pflanze1, Pflanze2, Pflanze3, Pflanze4, Pflanze5	5000	Gras
Berge-Steine	(HEIGHT>-1,1) AND (HEIGHT<-0,95)	Stein1, Stein2, Stein3	200	-

Tabelle 3.6: Selectors mit Ausdrücken

Änderungen

Der Vorteil gegenüber einer manuellen Spielwelterstellung besteht darin, dass die Spielwelt-Definition auf verschiedenen Abstraktionsebenen modifiziert und erweitert werden kann, ohne dass Schnittstellen oder Quelltext geändert werden müssen. In Abbildung 3.19 sind diese Ebenen dargestellt. Wird z. B. ein Fluss in eine Spielwelt gesetzt, so muss nur eine Metaebene geändert werden und die Spielwelt kann mit den geänderten Daten neuberechnet werden, ohne dass die Konsistenz verletzt wird und sich z. B. Bäume oder Häuser im neu erstellten Fluss befinden.

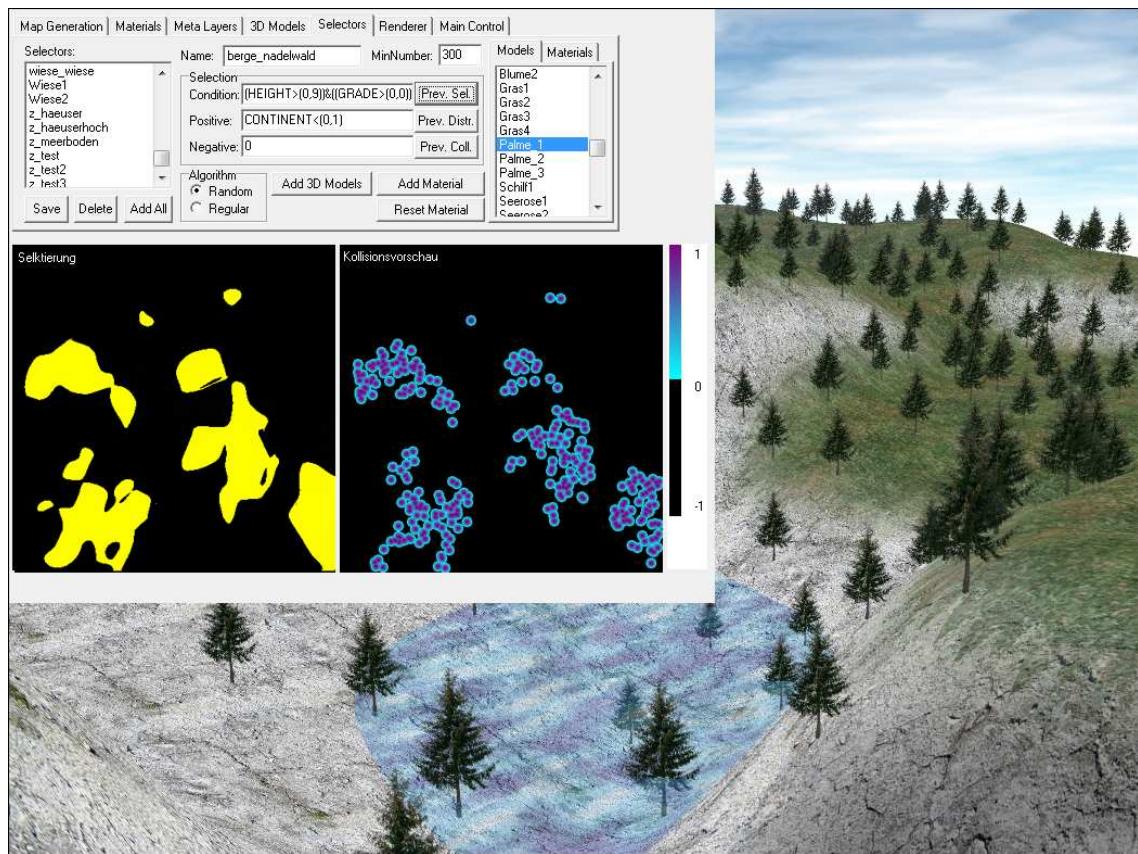


Abbildung 3.17: Bedienschnittstelle mit Selector-Definition und Vorschau

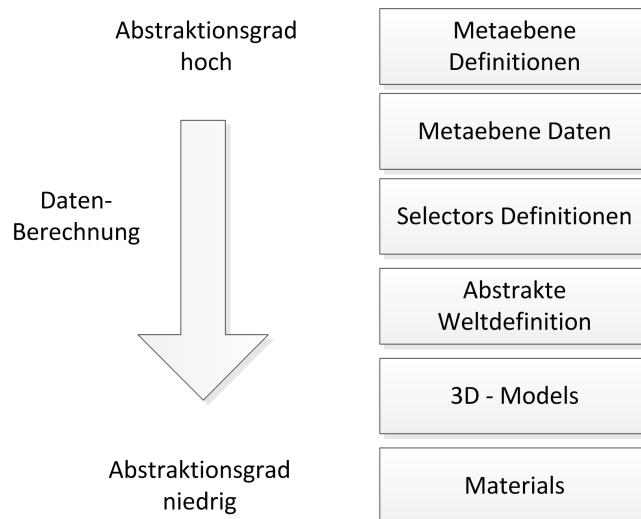


Abbildung 3.19: Abstraktionsebenen zur Modifikation von Spielwelt-Definition und Objekten

3.3.6 Parametrisierung und Zufall

Damit trotz des hohen Grades an Automatisierung und zufälliger Erzeugung bestimmte Parameter der Spielwelt kontrolliert werden können⁷, wurden geeignete Parameter wie z. B. Schroffheit der Berge, Wasser etc. festgelegt, die im Spielweltgenerator eingestellt werden können, diese können einen Wert von 0 bis 1 annehmen und sind in Abbildung 3.20 dargestellt und erklärt. Die Werte der Parameter können mit vorgesetztem \$ innerhalb von Ausdrücken als Variablen⁸ verwendet werden: Für die Ebene TEMPERATURE ist z. B. folgender Ausdruck definiert:

```
TEMPERATURE := transform[CONTINENT;0;50;1;20;2;25;3;-10;4;35]+($5)
```

`transform` bezeichnet eine Funktion, die Werte aus der Ebene `CONTINENT` in ungefähre Durchschnittstemperaturwerte umrechnet, d. h. Kontinent-0 zu 50°, Kontinent-1 zu 20° usw. (\$5) definiert den 5. Parameter, der sich über die Bedienschnittstelle einstellen lässt, als Variable der Funktion. Bei der Generierung einer Spielwelt wird vom Interpreter an dieser Stelle der vom Benutzer eingestellte Wert eingesetzt. So können die `TEMPERATURE`-Metadaten kombiniert aus den `CONTINENT`-Metadaten und der Parametereinstellung erzeugt werden.

Parameterbezeichnung	Variablen-Nr.	Bedeutung
<i>Mountain Count</i>	\$1	Bestimmt die Anzahl von Bergen und Hügeln, die bei der Generierung zufällig in der Heightmap platziert werden.
<i>Mountain Height</i>	\$2	Skalierungsfaktor der Heightmap, d. h. der Parameter skaliert die 3D-Koordinaten der Grundfläche in vertikaler Richtung.
<i>Mountain Roughness</i>	\$3	Dieser Parameter bestimmt die Schroffheit von Bergen, d. h. er stellt die Stärke der Erosion (vgl. [31], S. 5 bis 15) ein, mit der die Heightmap erstellt wird.
<i>Water</i>	\$4	Legt die Höhe des Wasserspiegels fest. Je höher das Wasser steht, desto weniger Landfläche gibt es.
<i>Climate</i>	\$5	Bestimmt Klimaeigenschaften von kalt und feucht (bei Minimalwert) bis warm und trocken (bei Maximalwert).

Abbildung 3.20: Eingabeparameter

Die anderen Parameter dienen ebenfalls als variable Eingabewerte, die drei Parameter *Mountain Count*, *Mountain Height* und *Mountain Roughness* werden dem

⁷vgl. Abbildung 3.1

⁸die in 3.2.3 definierte Grammatik wird dadurch um Variablen der Form '\$' [1-5] erweitert

Algorithmus zur Heightmap-Generierung übergeben⁹ und der Parameter *Water* legt die Höhe des Wasserspiegels fest. Die Parameter können über die Bedienschnittstelle manuell eingestellt werden oder zufällig variiert werden, in Abbildung 3.21 ist das Eingabefenster dazu dargestellt.

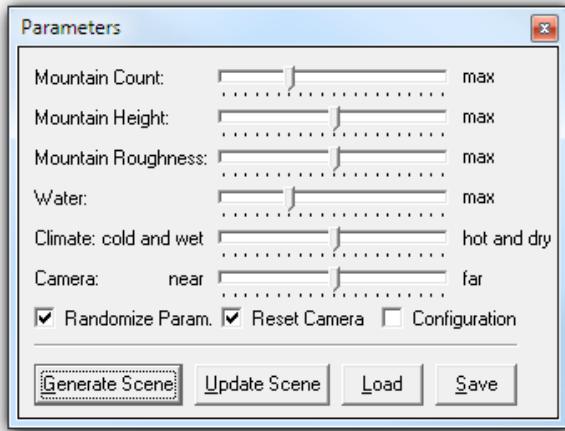


Abbildung 3.21: Bedienschnittstelle zur Einstellung der Parameter

3.3.7 Beispielkonfiguration

Anhand einer Konfiguration des Metamodells, das aus Definitionen von Metaebenen besteht und einer abstrakte Weltdefinition, die aus Definitionen von Selectors und zugeordneten 3D-Models und Materialien besteht, lässt sich anhand der Eingabe-parameter und anhand von Zufallswerten die Spielwelt automatisch generieren. In diesem Abschnitt wird eine Beispielkonfiguration aus Metamodell und abstrakter Weltdefinition demonstriert, die automatisch Zonen mit verschiedenen Landschaftsformen erzeugt.

Metamodell

Die Beispielkonfiguration besteht aus sechs Metaebenen, die in Abbildung 3.22 gezeigt sind. Die Ebenen `CONTINENT` und `HEIGHT` erzeugen die Werte des Datenfelds unter Benutzung der Funktion `random`. Diese Funktion erzeugt Zufallswerte auf Basis von Voronoi-Diagrammen ([27], S. 209 bis 215). Über die Parameter lassen sich Granularität und Anzahl der Zonen einstellen. So kann eine einfache, zufällige Aufteilung der Fläche z. B. auf eine bestimmte Anzahl von Kontinenten erzeugt werden. Die Funktion `random` unterstützt weitere Parameter, die die Erzeugung einer zufälligen Heightmap beeinflussen, wie beispielsweise der in Abbildung 3.22 bei der

⁹siehe Definitionen der Metaebenen in Abbildung 3.22

Definition der Ebene `HEIGHT` mit der Variable `$3` für *Mountain Roughness* verknüpfte Parameter, über den die Stärke von Erosionsfiltern (vgl. [31], S. 5 bis 15) bestimmt wird. Die restlichen Parameter dienen der Konfiguration technischer Details wie z. B. der Einstellung eines Glättungsfaktors, auf die hier nicht weiter eingegangen wird.

Die Definitionen der Metaebenen beinhalten durch Benutzung der Funktion `random` Zufallswerte als Eingabequelle. Zum Verständnis, wie sich Zufall und Eingabewerte auf die berechneten Metadaten auswirken, ist in Abbildung 3.23 der Fluss von Daten zwischen den Ebenen, von Zufallswerten und von den Eingabeparametern der Bedienschnittstelle dargestellt.

Abstrakte Weltdefinition

In Abbildung 3.24 ist die abstrakte Weltdefinition der Beispielkonfiguration illustriert, sie besteht aus einer Liste von Selectors und zugeordneten Materials bzw. 3D-Models. Auf deren Basis wurden durch zufällige Variation der Parameter und durch Zufallsalgorithmen generierte Metadaten der Ebenen `CONTINENT` und `HEIGHT` die in Abbildung 3.23, 3.24 und 3.25 dargestellten Spielwelten generiert.

Wenn man bedenkt, dass nur wenige elementare Faktoren variiert werden, so ist die Vielfalt der generierten Szenen erstaunlich. Sie lässt sich anhand der drei Beispiele nur ausschnittsweise darstellen.

Aufgrund der unidirekionalen Abhängigkeiten der Metaebenen können nun zu einer komplett zufällig erstellten Szene einzelne Parameter variiert werden und die Szene kann neu berechnet werden, ohne dass die Konsistenz verletzt wird: Die Änderungen wirken sich nur isoliert auf die von den Parametern abhängigen Faktoren aus.

Über die Funktion *Update Scene*¹⁰ der Bedienschnittstelle lässt sich nach Änderung von Parametern eine neue Szene generieren, sodass zufällig erzeugte Merkmale teilweise erhalten bleiben. Dadurch, dass der Anker für die verwendeten Pseudozufallszahlen (engl. *Random Seed*) nicht geändert wird, werden bei der Neuberechnung der Spielwelt dieselben Zufallszahlen wie vor der Änderung erzeugt, was in dem folgenden Beispiel dazu führt, dass Änderungen an verschiedenen Parametern auf die Szene übertragen werden können, ohne dass sich z. B. die zufällig erzeugten Positionen von Bergen verändern.

In Abbildung 3.28 ist ein Beispiel dargestellt: Bei derselben Szene wurde beim linken Bild der Parameter *Climate* auf ein Wert für kälteres Klima gesetzt und beim rechten Bild der Parameter *Mountain Height* erhöht. Die Änderungen wirken sich innerhalb der Definitionen des Metamodells aus. Die Hauptcharakteristik der Szene bezüglich der anderen Parameter bleibt jeweils erhalten.

¹⁰vgl. Abbildung 3.21

3.3.8 Performance

Die Software ist in Delphi implementiert und nutzt für die grafische Darstellung OpenGL. Die Beispielszenen basieren auf Heightmaps, Metaebenen und Meshs mit einer Auflösung von 400x400 Werten und beinhalten ungefähr 5000 Landschaftsobjekte. Die Generierung einer Szene nimmt wenige Sekunden in Anspruch, sodass die 3D-Visualisierung der erstellten Szene mit nur geringer Verzögerung betrachtet werden kann, während die Szene in Echtzeit gerendert wird und der Kamerawinkel frei gewählt werden kann.

In Tabelle 3.7 ist die Generierungsdauer einer Spielwelt auf demselben System bei Variation der Auflösung der Datenfelder und der Anzahl der Spielwelt-Objekte verglichen.

Generierungsdauer in Sekunden	Auflösung der Datenfelder	Anzahl der Spielwelt-Objekte
1,33	200x200	4640
2,29	200x200	16686
3,41	400x400	5386
5,88	400x400	18229
12,07	800x800	5095
13,63	800x800	17067
51,08	1600x1600	3182
52,88	1600x1600	15389

Tabelle 3.7: Messungen der Generierungsdauer

Die Werte zeigen, dass die Generierungsdauer bei einer Erhöhung der Auflösung der Datenfelder deutlich ansteigt, was durch die größere Datenmenge und dadurch höhere Anzahl von Rechenoperationen auf den Daten bedingt wird. Es bestehen daher u. a. Möglichkeiten der technischen Optimierung, um die Generierungsdauer zu minimieren:

- durch die Ausrichtung der Datenverarbeitung der CPU auf Mehrprozessorkerne
- durch die Ausführung von Rechenoperationen auf den Metadaten durch Grafikkartenprozessoren (engl. *Graphics Processing Unit*, Abk. *GPU*), z. B. durch die *Compute Unified Device Architecture (CUDA)* Technik (vgl. [13])

Eine geringe Verzögerungszeit begünstigt eine interaktive Bedienung, sodass die Auswirkungen von Änderungen an Eingabeparametern oder Definitionen auf die Spielwelt schneller und besser überblickt werden können.

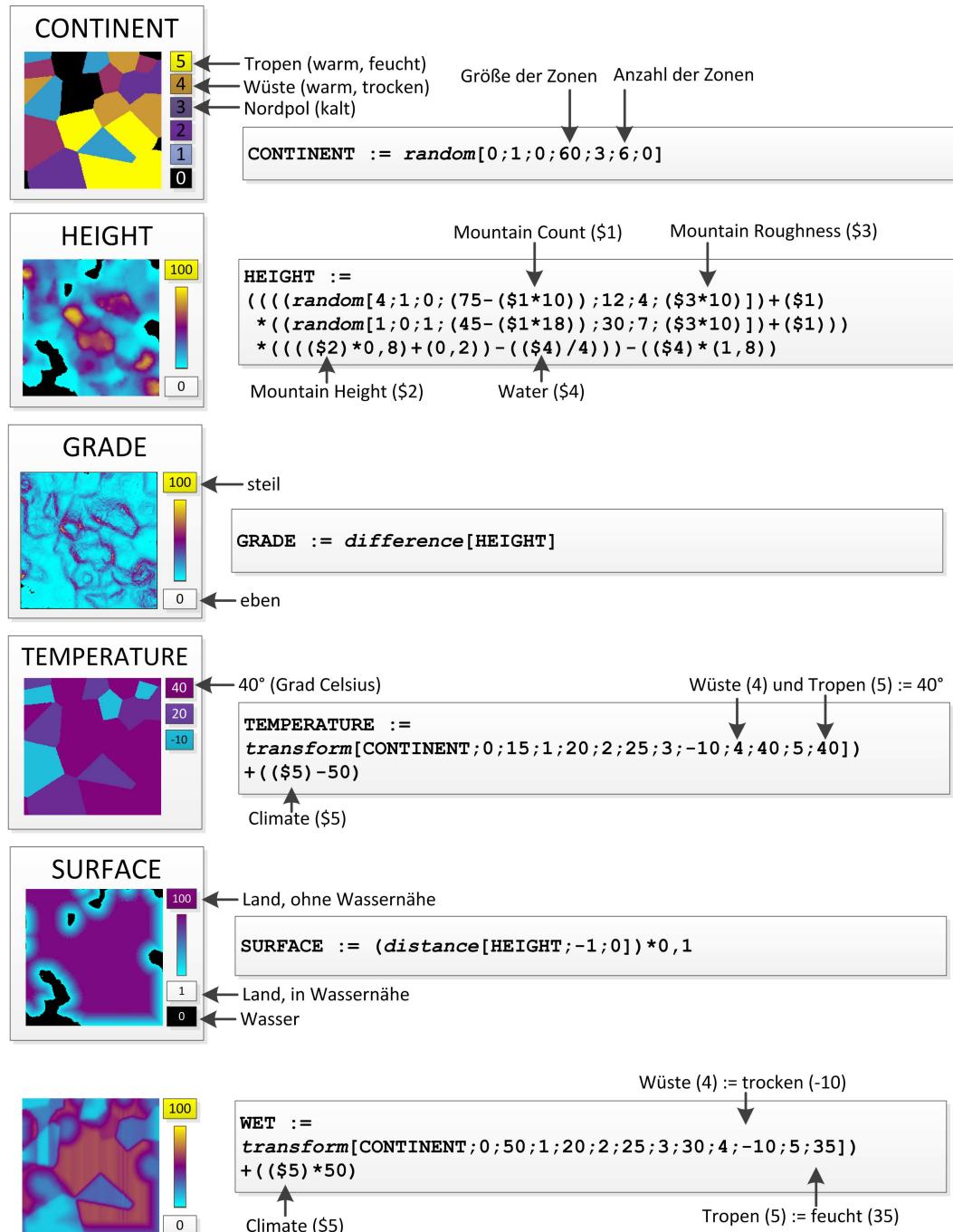


Abbildung 3.22: Metamodell der Beispielkonfiguration

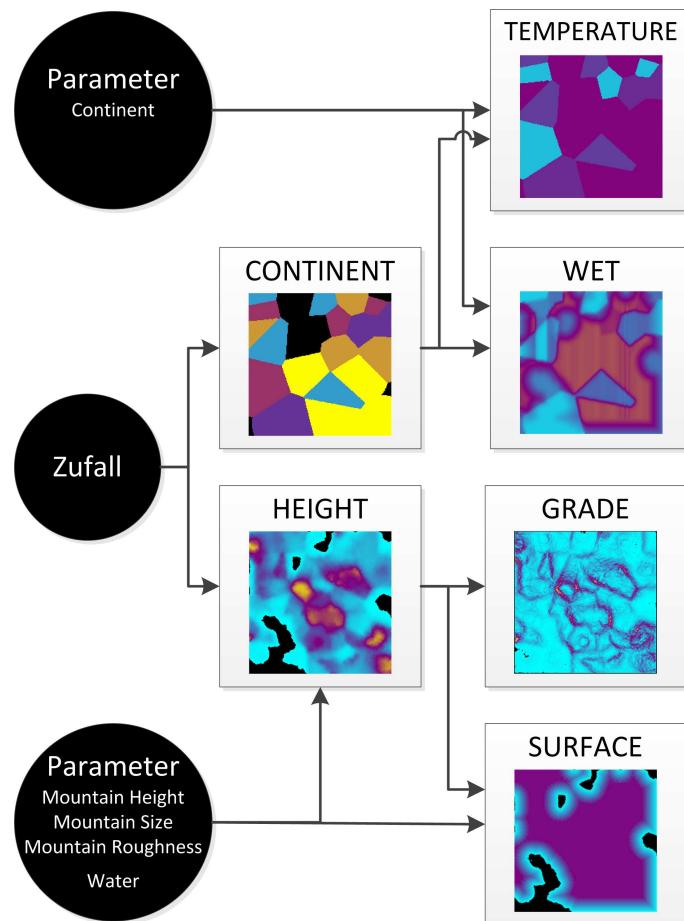


Abbildung 3.23: Metaebenen mit Eingangsquellen von Daten in der Beispielkonfiguration

Selectors – Materials		Selectors – 3D-Models		
Material	Selektierungsausdruck	3D-Models	Selektierungsausdruck	
Grund-material	 (TRUE)	Laubbäume	(HEIGHT<0,1) AND (HEIGHT>-0,8) AND (GRADE<-0,75) AND (TEMPERATURE<0)	
Fels		(GRADE>(-0,4)) OR (HEIGHT>(0,01))	Laubbäume 2	(HEIGHT<0,4) AND (HEIGHT>-0,3) AND (GRADE<-0,4) AND (TEMPERATURE>0,4)
Gras		(HEIGHT<-0,1) AND (GRADE<-0,8) AND (WET>-0,5)	Wiese	(HEIGHT<0,1) AND (HEIGHT>-0,8) AND (GRADE<-0,75)
Sand		(HEIGHT<-0,8) AND (HEIGHT>-1,1) AND (GRADE<-0,8) AND (TEMPERATURE>0,1)	Palmen	(HEIGHT<-0,8) AND (HEIGHT>-0,9) AND (GRADE<-0,8) AND (TEMPERATURE>0,4)
Schnee		(HEIGHT>(0,2+(\$5)) AND (GRADE<-0,6))	Tannen	(HEIGHT>-0,5) AND (GRADE<-0,6) AND (HEIGHT<0,2) AND (SURFACE>0)
Steine		(HEIGHT<-0,98) AND (HEIGHT>-1,05) AND (GRADE<-0,6))	Steine	(HEIGHT<-0,98) AND (HEIGHT>-1,05) AND (GRADE<-0,6))
Boden trocken		(HEIGHT<-0,1) AND (GRADE<-0,8) AND (WET<-0,7) AND (TEMPERATURE<0,2))	Schilf	(SURFACE<0,1) AND (HEIGHT>-1,05) AND (GRADE<-0,6))
Wasser		(SURFACE=0)	Seerosen	(SURFACE=0) AND (HEIGHT>-1,1) AND (GRADE<-0,6))

Abbildung 3.24: Abstrakte Weltdefinition der Beispielkonfiguration



Abbildung 3.25: Zufällig generierte Szene 1



Abbildung 3.26: Zufällig generierte Szene 2



Abbildung 3.27: Zufällig generierte Szene 3

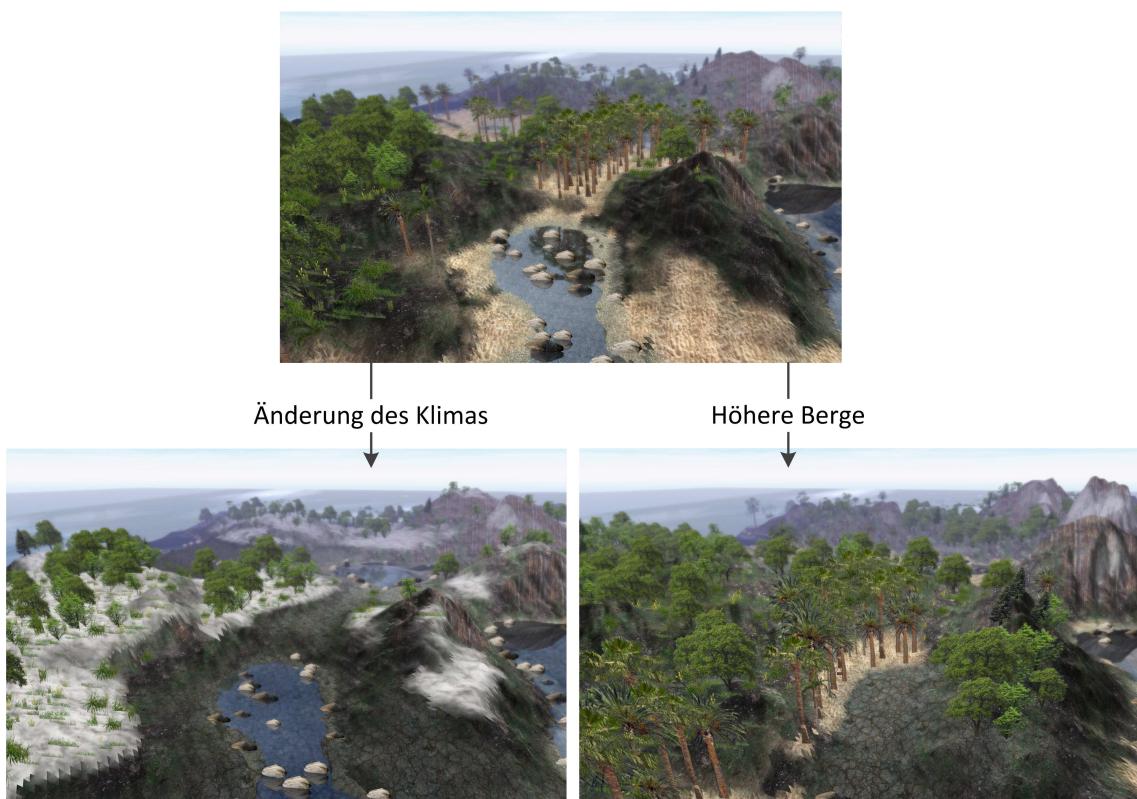


Abbildung 3.28: Modifikation von Parametern an einer Szene

4 Auswertung

In diesem Teil wird das Projekt hinsichtlich Methoden und Werkzeugen der Softwaretechnik und der entworfenen Architektur ausgewertet. Zudem werden Erweiterungsansätze genannt und die Ergebnisse evaluiert.

4.1 Einsatz von Softwaretechnik

Bei der Implementierung wurden verschiedene Techniken und Entwurfsmuster zur Verwaltung und Darstellung der Daten verwendet, die für Architekturen von Spiele- und Grafikengines üblich sind (vgl. [6]). Im Folgenden wird auf die Besonderheiten der eingesetzten Softwaretechnik eingegangen.

Das Problem bei der Modularisierung des Spielweltgenerators ist die Komplexität, die durch eine Modellierung der Beziehungen zwischen dem abstrakten Spielweltmodell und den konkreten 3D-Modellen und Materialien durch strikte Schnittstellen entstehen würde: Die Definitionen der Metaebenen und die Metadaten sollen jedoch einfach austauschbar sein, ebenso 3D-Modelle und Materialien.

Beim Generierungsalgorithmus weist der Schritt, bei dem die Spielwelt-Objekte in der Spielwelt erstellt werden, ein Maximum an Kohäsion auf, wie Abbildung 4.1 zeigt.

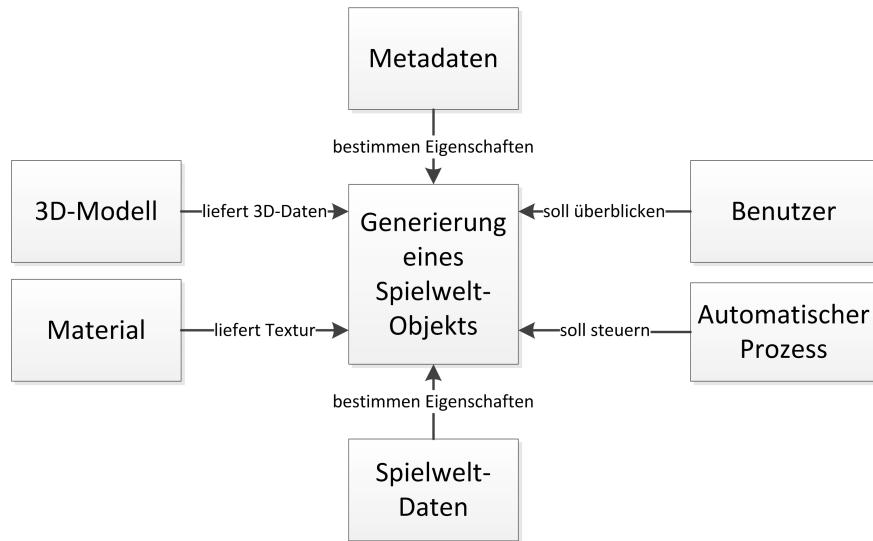


Abbildung 4.1: Kohäsion bei der Generierung eines Spielwelt-Objekts

Alle Komponenten haben Einfluss auf die Generierung von Spielwelt-Objekten: Die Daten der 3D-Models und Materialien werden als Objekte direkt in der Spielwelt instantiiert bzw. referenziert. Der Benutzer möchte festlegen, wie die Spielwelt aussieht. Das Metamodell soll den Automatisierungsgrad zur Generierung maximieren.

Gleichzeitig ist es wichtig, die Kopplung der 3D-Modelle und Materialien an das Metamodell so flexibel wie nur möglich zu gestalten, um nicht die Möglichkeit von beliebigen Beziehungen zwischen Spielwelt-Objekten und den in den Metadaten enthaltenen semantischen Daten zu beschränken: Ein Spielwelt-Objekt soll jedes Phänomen darstellen können und mit beliebigen Metadaten verknüpft werden können.

Die abstrakte Weltdefinition kombiniert die Entwurfsmuster *Interpreter* und *Command* (vgl. [20]) und verlagert die Komposition von Selectors und 3D-Models in den Zuständigkeitsbereich des Benutzers, der die abstrakte Weltdefinition erstellt. Dadurch löst die Architektur den Konflikt zwischen Kohäsion und Kopplung an dieser Stelle. In Abbildung 4.2 ist dies dargestellt.

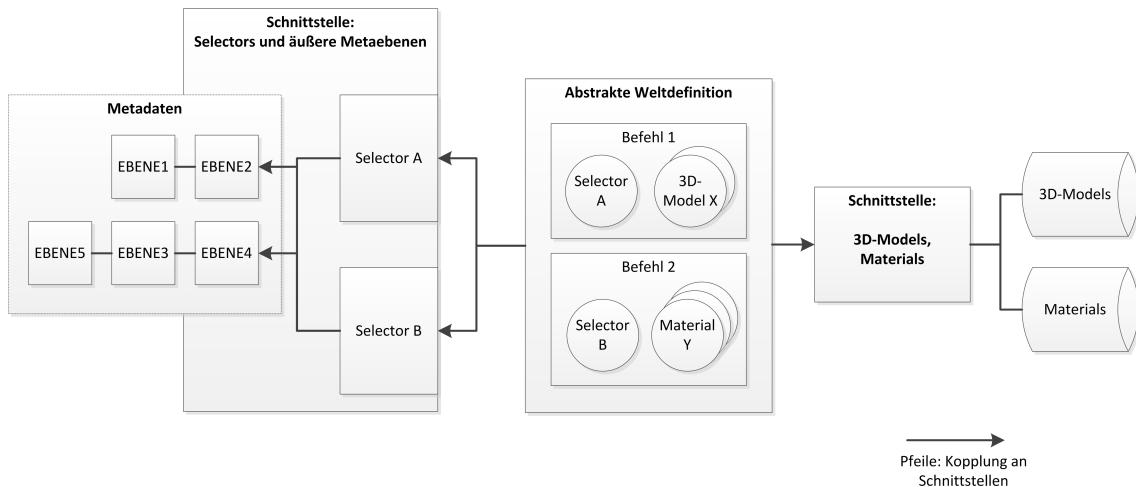


Abbildung 4.2: Abstrakte Weltdefinition mit Kopplung

4.2 Datentrennung in der Architektur

Die Architektur des Spielweltgenerators kapselt die Metadaten und die darauf definierten Ausdrücke von den materiellen Spielwelt-Daten ab und reduziert die Schnittstelle: Metadaten sind nur über die Befehle der abstrakten Weltdefinition an die Spielwelt-Daten gekoppelt, indem in den Befehlen Selectors (räumliche Selektierungen auf *Metadaten*) mit 3D-Modellen und Materialien (Daten, die in die *Spielwelt* gesetzt werden) kombiniert werden.

Der Vorteil dieser Minimierung der Kopplung zwischen Metadaten und Spielwelt-Daten ist eine klare Trennung der Semantik der beiden Datenmodelle. Während die

semantisch reichhaltigen Metadaten die Generierung der Spielwelt-Daten bestimmen, sind die erzeugten Spielwelt-Daten Rohdaten aus einfachen Objekten, Grafiken und 3D-Koordinaten. Das dem Muster *Abstract Factory* ([20], S. 101) zugrundeliegende Prinzip, bei der Herstellung von Produkten zusätzliche Funktionalität, Komplexität und Eigenschaften im Herstellungsprozess und in den konkreten Produkten zu erreichen, die vom Benutzer der abstrakten Schnittstelle entkoppelt sind, sodass sie variiert werden können, wird hier angewandt. Semantische Daten, die den Generierungsprozess und die Ausprägung der Spielwelt bestimmen, werden entkoppelt von den konkreten Spielwelt-Daten, die als Rohdaten produziert werden und vom Anwender benutzt werden. Je abstrakter, einfacher und universeller die Schnittstelle zur Generierung einer Spielwelt ist, desto stärker kann die Logik des Generierungsprozesses, die sich im hergestellten Produkt, der konkreten, generierten Spielwelt wiederfindet, von den Anwendungen, digitalen Spielen, entkoppelt werden. Je stärker diese Entkopplung ist, desto mehr können Komplexität, Umfang und Effizienz des Generierungsprozesses und der Architektur des Spielweltgenerators erweitert und verbessert werden, ohne dass die Schnittstelle zur Anwendung, dem digitalen Spiel, von Änderungen betroffen ist.

Die Semantik der Spielwelt-Daten einer Spielwelt, die vom Spielweltgenerator als konkretes Produkt zur Verwendung in einem digitalen Spiel bzw. Spiel-Prototyp erstellt wird, wird von dem Gameplay des jeweiligen digitalen Spiels bestimmt und dieses wiederum vom Game Design. Bei Entwurf und semantischer Ausgestaltung der Spielwelt-Objekte, die sich direkt oder indirekt im Gameplay wiederfinden, sollte daher eine aufs Gameplay bezogene Perspektive vorherrschen, die sich möglichst direkt an den Zielen und Dokumenten des Game Design orientiert, im Gegensatz zu dem Bau des Spielweltgenerators, der leistungsfähig produzieren soll.

4.3 Erweiterungsmöglichkeiten

In diesem Abschnitt werden in Hinblick auf die Anwendungsmöglichkeiten Ansätze und Überlegungen für Erweiterungen der Implementierung und des Modells formuliert.

4.3.1 Erweiterungen der Implementierung

Die Arbeit befasst sich eingehend mit Modell und Architektur. Die Umsetzung wurde auf wesentliche Funktionen beschränkt, um eine gute Qualität und Skalierbarkeit zu erreichen. Im Folgenden werden Möglichkeiten genannt, welche die Funktionalität quantitativ erweitern und technische Details verbessern.

Import und Export

Eine Implementierung des Exports der Spielwelt-Daten und des Imports von 3D-Models in üblichen 3D-Dateiformaten bietet die Möglichkeit, den Spielweltgenerator an aufwändige, professionelle 3D-Grafiksoftware, Grafik- und Spieleengines anzubinden.

Interaktivität

Die Bedienschnittstelle lässt sich für ein höheres Maß an Komfort und Interaktionsmöglichkeit erweitern, indem selektierte Bereiche als Vorschau, bevor Objekte generiert werden, nicht nur als 2D-Visualisierung erstellt, sondern direkt in die 3D-Szene integriert visualisiert werden.

Die Möglichkeit, sich über die Software mit einer Spielfigur in der Spielwelt bewegen und mit Objekten interagieren zu können, z. B. durch physikalische Kollision, würde ebenfalls die Interaktivität der Bedienung steigern.

Optimierung und Details des Rendering

Die Performance des Rendering wurde nur rudimentär optimiert, um den Zweck einer einfachen 3D-Vorschau der Spielwelt-Daten zu erreichen, sodass ein realistischer Eindruck der generierten Spielwelt gewonnen werden kann. Durch den Einsatz von Techniken, die von aufwändigen Grafikengines eingesetzt werden oder eine Anbindung an andere Frameworks zur 3D-Darstellung, lassen sich die Performance und auch der Umfang an Grafikeffekten steigern.

Dynamische Generierung

Das Hinzufügen einer Funktionalität der dynamischen Generierung, die neue Spielweltabschnitte an den Grenzen der erstellen Spielwelt generiert, sobald sich der Spieler in deren Nähe befindet, würde es erlauben, endlose Spielwelten zu generieren. Auch eine Adoptionsfähigkeit der Spielwelt an nichtlineare Spielabläufe ließe sich auf Basis von dynamischer Generierung zur Laufzeit eines Spiels realisieren.

4.3.2 Erweiterungen des Modells

Im Folgenden werden qualitative Erweiterungsmöglichkeiten des Modells genannt, die das Ziel verfolgen, den Automatisierungsgrad zu steigern und die Definition komplexerer Ausdrücke effizienter zu gestalten.

Höhere Komplexität

Durch die Möglichkeit, neue Ebenen im Metamodell zu definieren und den Zusatz von Operatoren bzw. Filtern, die in mehreren Stufen auf den Metadaten operieren, lassen sich auch speziellere Zusammenhänge definieren. Abbildung 4.3 zeigt ein Beispiel, bei dem die Kreuzungen eines Straßennetzes mit Ampeln ausgestattet werden, indem aus einer Ebene mit den Straßendaten durch eine einfache Hough-Transformation [24] die Ecken der Kreuzungen gefiltert wurden. Das Straßennetz ist gelb dargestellt und die für die Ampeln zulässigen Bereiche blau.

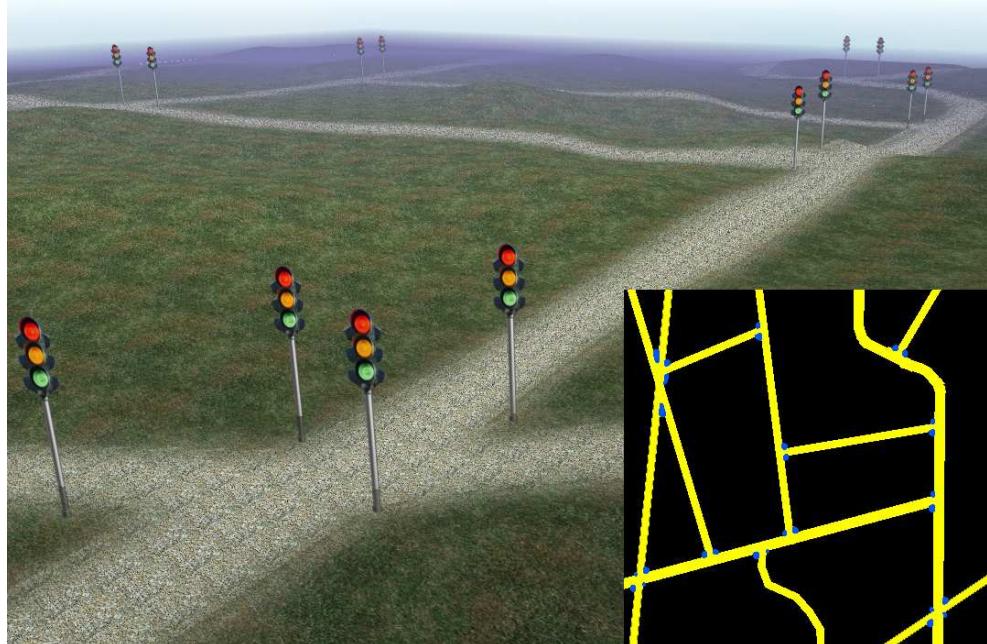


Abbildung 4.3: Ampeln an Kreuzungen

Die Funktionalität und Intelligenz ist hierbei in den Filter bzw. einen Operator des Interpreters gekapselt, sodass die Komplexität des Algorithmus, den der Operator verwendet, sich nicht auf die Architektur auswirkt.

Ein Nachteil ist jedoch, dass mit zunehmender Anzahl von Metaebenen und zunehmender Komplexität der Ausdrücke die Benutzung schwieriger wird, da die Herleitung der einzelnen Daten immer schlechter überschaut werden kann. Dies kann zum einen durch Dokumentation und Spezifikation der Semantik einzelner Metadaten verbessert werden, andererseits wäre ab einer gewissen Datenmenge auch eine Automatisierung der Selektierungsausdrücke durch *Clusteranalyse* (vgl.[3]) denkbar, indem aus Geoinformationssystemen oder Satellitenbildern die räumlichen Bereiche von Vorkommen eines bestimmten Phänomens im multidimensionalen Raum des Metamodells extrahiert werden. Die Selectors können dann die räumlichen Bereiche statt durch Auswertung eines manuellen Ausdrucks anhand des Umfeldes von *Clustern* bzw. *Punktwolken* selektieren.

Abhängigkeiten von Metamodell und Spielwelt-Daten

Die Generierung der Spielwelt-Objekte ist durch die Selektierungsausdrücke auf dem Metamodell von den Metadaten und u. a. durch die Vermeidung von Kollision mit anderen Spielwelt-Objekten von den Spielwelt-Daten abhängig. Somit beeinflussen die Metadaten unidirektional Spielwelt-Daten. Sinnvoll wäre es, das Modell um eine mögliche Definition von Metadaten aus Spielwelt-Daten zu erweitern. Werden beispielsweise Häuser platziert, wäre eine Ebene mit Metadaten, die Werte für die Bevölkerungsdichte beinhaltet, als Basis für die Generierung weiterer Objekte, z. B. für Müll, der herumliegt, hilfreich.

Für eine entsprechende Erweiterung müsste der Interpreter um die Möglichkeit, Werte nicht nur aus Metadaten, sondern auch aus den Spielwelt-Daten, lesen zu können, erweitert werden. Sofern die Gesamtheit der durch Ausdrücke definierten Abhängigkeiten kreisfrei bleibt, lassen sich dann Metaebenen auch unter Einbeziehung von Spielwelt-Daten definieren, wie Abbildung 4.4 an einem Beispiel zeigt, bei dem erst Häuser in der Spielwelt generiert werden und dann daraus resultierende Daten der Spielwelt zur Berechnung von Metaebenen verwendet werden. Die funktionale Semantik innerhalb des Metamodells wird dadurch unübersichtlicher und die bislang vernachlässigte Befehlsreihenfolge spielt dadurch eine entscheidende Rolle beim Generierungsprozess.

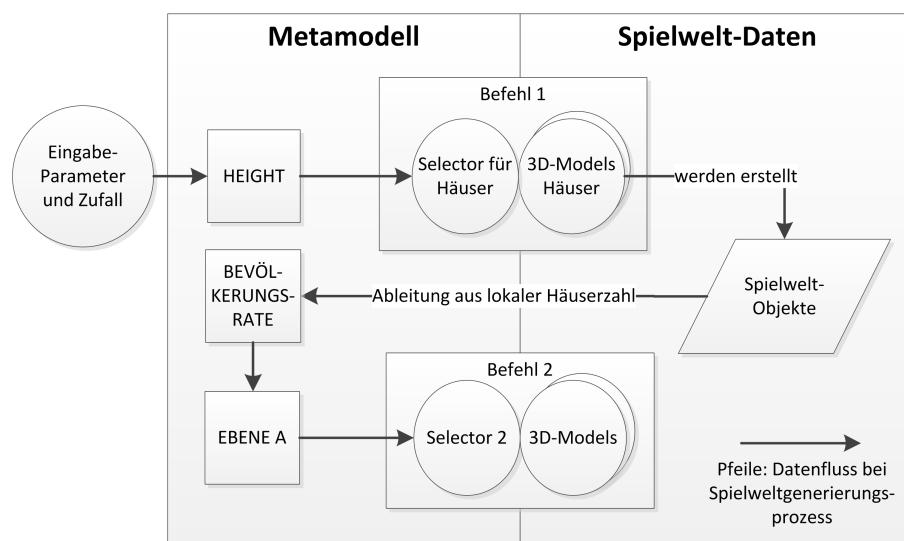


Abbildung 4.4: Informationsrückfluss von den Spielwelt-Daten zum Metamodell

Ab einem gewissen Detailgrad der Modellierung von Faktoren, welche die Generierung von Spielwelt-Objekten beeinflussen, werden solche Rückflüsse von Informationen zum Metamodell jedoch nötig. Eine Unterteilung des Metamodells in mehrere feste Ebenengruppen, sodass die Metaebenen und Selectors in eine hierarchische Struktur gebracht werden, die in Form von einer Halbordnung die zeitliche Ablauf-

reihenfolge der jeweiligen Datenberechnung beim Generierungsprozess einschränkt, wäre eine Grundlage, solche Rückflüsse zum Metamodell kontrollieren zu können.

Durch die deklarative Definition der Selektierungsausdrücke ergibt sich das Problem, dass bei konkurrierenden Selectors, welche überlappende räumliche Bereiche selektieren, der zuerst ausgeführte Befehl diese Ressourcen belegen kann und dadurch gegen spätere Befehle gewinnt. In der Implementierung hat bei der Platzierung von Spielwelt-Objekten der Erste den Vorrang, denn einmal platzierte Objekte werden nicht wieder entfernt. Bei Materialien ist es umgekehrt, eine Materialzuordnung wird durch spätere Befehle mit anderen Materialzuweisungen desselben Bereichs überschrieben. Eine Behandlung solcher Konflikte zur Bestimmung der Gewinner und Verlierer unabhängig von der zeitlichen Ausführung, durch z. B. eine Prioritätenvergabe, ist eine Grundlage, diese Problematik kontrollieren zu können (vgl. [41], S. 356).

4.4 Ergebnis und Fazit

Im folgenden Abschnitt wird der Nutzen im Umfeld der Spieleentwicklung diskutiert und ein Fazit gezogen.

4.4.1 Ergebnis

Mit dem entwickelten Spielweltgenerator lassen sich Spielwelten bis zu einem gewissen Komplexitätsgrad einfach und effizient generieren und darstellen. Modellierung und Entwurf auf sehr hoher Abstraktionsebene haben einen großen Teil der Arbeit in Anspruch genommen, sich aber als leistungsfähig und erweiterbar herausgestellt: Die Definition der domänenspezifischen Sprache ist im Verhältnis zum Aufwand besonders nützlich, da über die Bedienschnittstelle sehr schnell Zusammensetzungen von Objekten definiert und automatisch platziert werden können. Im Vorhinein bestanden Zweifel an einer guten Funktionalität aufgrund der umfassenden Komplexitätsreduktion, u. a. durch Einschränkung des Metamodells auf Kreisfreiheit der Abhängigkeiten der Ebenen und der sehr abstrakten Semantik von Daten und Definitionen. Neue Sprachen müssen erst einmal erlernt werden und durch die Sprache sind umfassende Änderungen mit weitreichender Wirkung möglich, z. B. der Ebenendefinitionen des Metamodells. Deshalb bestand das Risiko, dass die flexible Architektur zu nicht vorhergesehenen Problemen oder die Vereinfachung des Modells zu einer komplizierten Bedienung führt: Es hat sich aber gezeigt, dass die Einfachheit der Sprache, die aufgrund der Orientierung an mathematischen und logischen Ausdrücken intuitiv verständlich ist, besonders in Kombination mit der interaktiven Bedienbarkeit, die dem Benutzer sofort eine Visualisierung der Semantik des eingegebenen Ausdrucks vermittelt, auch zu einer einfachen Bedienbarkeit führt. Selbst

bei komplexeren und zusammengesetzten Ausdrücken ist die vom Ausdruck gelieferte Visualisierung aussagekräftig, sodass sich das Metamodell vom Benutzer effizient experimentell verfeinern und anpassen lässt.

Begrenzt sind Modell und Software einerseits in der Komplexität, da sie keine Kreise in den Abhängigkeiten der Definitionen der Metaebenen zulassen, was für die Modellierung von Wechselwirkungen sinnvoll wäre: Solche lassen sich nicht auf der Definitionsebene darstellen, sondern können nur durch schrittweise Berechnung und manuelle Speicherung auf konkreten Daten angewandt werden.

Quantitativ stellt die Menge der zum Testen verwendeten 3D-Modelle, die hauptsächlich aus einfachen Repräsentationen von Pflanzen und Landschaftsobjekten bestehen, nur ein Beispiel dar, welches für Spielwelten bestimmter Spiele, besonders solcher, die keine *Virtual Reality* anstreben, nur geringe bis gar keine Anwendungsmöglichkeiten bietet.

Einige Anwendungsmöglichkeiten des erstellten Spielweltgenerators, der Architektur bzw. des Modells, sind:

- *Sketching und Artwork*: Im frühen Entwicklungsstadium von Spielen lassen sich effizient Grafikmedien anhand von ersten Dokumenten oder Ideen zum Spiel erstellen und somit visuell ausdrücken, indem auf hoher Abstraktionsebene mit Wirkung auf alle niedrigeren Ebenen¹ Redundanzen vermieden werden und effektiv auf bereits erstellte Medien zurückgegriffen werden kann: Statt für eine einzelne Szene eine Skizze anfertigen zu müssen, können die Elemente für die Platzierung in Form von einfachen Ausdrücken beschrieben und Objekte effizient zusammengestellt werden. Da automatisch eine 3D-Spielwelt generiert und dargestellt wird, ist es möglich sich wie ein Fotograf in der Welt zu bewegen und aus beliebigen Kamerawinkeln Screenshots zu erzeugen: Die Software eignet sich als kreatives Tool zur Szenengestaltung.
- *Prototyping und experimentelles Game Design*: Die flexible Spielweltgestaltung bietet die Möglichkeit, experimentell Spielprinzipien und -aspekte zu testen, die den Aufbau größerer Spielwelten erfordern. Beispielsweise ist das Suchen und Sammeln bestimmter Gegenstände oft Teil von Rollenspielen und digitalen Spielen ähnlicher Genre. Solche Spiele profitieren in hohem Maße von der Immersion, die sie dem Spieler bieten. Wird diese durch Vorhersehbarkeit bzw. dadurch, dass der Spieler das Spiel *durchschaut*, gemindert, so sinkt der Spielspaß eklatant ab (vgl. [1], S. 4 bis 5). Wie bei Filmen und Büchern, deren Ende man bereits kennt, können Spannungs- und Überraschungsmomente und damit der Spielspaß abnehmen. Ein Einsammeln von Gegenständen, die nach einem offensichtlichen Prinzip in einem gewissen Umfeld platziert sind, kann daher leicht als lästig und unspektakulär empfunden werden. Im Metamodell kann Logik, die Ort und Eigenschaften der Gegenstände bestimmt, beschrieben werden. Da die Ebenen der Metadaten für den Spieler nicht sichtbar sind

¹vgl. Abbildung 3.19

und auch Zufallseinflüsse auf Metaebenen einbezogen werden können, kann so eine komplex erscheinende Logik, die innerhalb der 3D-Spielwelt Neugier und Interesse des Spielers weckt (vgl. [21]), einfacher definiert als vom Spieler durchschaut werden.

- *Verwendung der generierten Spielwelt in Spielen:* Die Spielwelt lässt sich auch direkt in Spielen einsetzen, wenn die Spielwelt-Daten in passende Formate der bei der jeweiligen Entwicklung benutzten Frameworks und Spieleengines konvertiert und je nach Anforderungen für die platzierten Objekte hochwertige 3D-Modelle eingesetzt werden.

4.4.2 Fazit

Das Projekt zeigt den Nutzen eines automatisierten Spielweltgenerators auf und die Realisierbarkeit anhand eines Top-down Entwurfs unter Verwendung von Prinzipien der Softwaretechnik auf hoher Abstraktionsebene. Entwurfsmuster und abstrakte Modelle haben geholfen, die Komplexität des Szenarios zu reduzieren und eine geeignete Architektur zu entwerfen. Der Entwurf einer eigenen Sprache und die Implementierung einer Interpreters haben sich dabei als besonders leistungsfähig herausgestellt.

5 Zusammenfassung und Ausblick

Beide Teile der Arbeit werden im Folgenden inhaltlich zusammengefasst und dann ein thematischer und zeitlicher Ausblick gegeben.

5.1 Zusammenfassung

Die Spieleindustrie hat sich mit ihrem rasanten Wachstum in den letzten Jahren zu einem aufstrebenden Segment der Unterhaltungsindustrie entwickelt. Digitale Spiele und deren Entwicklung sind auch zunehmend Gegenstand wissenschaftlicher Forschung. Die Bezeichnungen neuerdings eingeführter Studiengänge im Umfeld der Spieleentwicklung variieren ebenso wie die Ansichten, ob Spieleentwicklung eher als kreative Kunst oder technisches Handwerk zu verstehen ist. Die Themenbereiche der Spieleentwicklung Game Design, Softwareentwicklung und Mediendesign involvieren unterschiedliche Perspektiven auf das Spiel, die zu unterschiedlichen Bewertungen des Stellenwerts der Softwaretechnik in der Spieleentwicklung führen. Die Gegenüberstellung der Argumente und Analyse von Problemen und Widersprüchen in der Spieleentwicklung legt eine interdisziplinäre Sichtweise auf die Spieleentwicklung nahe. Sie stellt eine besondere Domäne der Softwareentwicklung dar, die sehr von Methoden und Werkzeugen der Softwaretechnik profitieren kann. Sie ist stark vom Gegenstand der Entwicklung, den digitalen Spielen, geprägt. Daraus folgt bei einem erweiterten Verständnis von Softwaretechnik im Rahmen der Spieleentwicklung eine Definition des *Game Engineering*, welches den Entwicklungsprozess nicht nur mit Blick auf die Software, sondern mit Blick auf das digitale Spiel optimiert.

Im Projekt wurde ein Spielweltgenerator entworfen und implementiert, wobei Methoden und Werkzeuge der Softwaretechnik wie Entwurfsmuster ebenso wie der aktuelle Wissenschaftsstand bezüglich Terraingenerierung und virtueller Welten mit praktischen Techniken der Spieleentwicklung und Computergrafik kombiniert wurden. Dabei wurde beim Modell- und Architekturentwurf besonderer Wert auf die domänenspezifischen Anforderungen gelegt, welche die Flexibilität und Erweiterbarkeit in den Vordergrund stellen. Die Architektur des Spielweltgenerators bietet auf hohem Abstraktionsniveau eine deutliche Komplexitätsreduktion, sodass sie leichter zur Generierung spezieller Spielwelten angepasst werden kann. Der Spielweltgenerator erstellt anhand nur weniger Parameter durch Kombination von Zufallseinflüssen und Generierungsregeln automatisch Spielwelten und bietet durch diese Aufwandsminimierung ein praktisches Hilfsmittel zur Effizienzsteigerung beim Prototyping und experimentellen Game Design in der Spieleentwicklung.

5.2 Ausblick

Die Zahl der neuen Studiengänge im Bereich der Spieleentwicklung hat im letzten Jahr weiterhin zugenommen. Damit hat sich ein Trend weiter fortgesetzt, sodass mittlerweile nicht nur private Hochschulen, sondern auch Hochschulen mit staatlicher Trägerschaft Studiengänge in dem Bereich anbieten, z. B. die Technische Universität München den Studiengang *Games Engineering* und die Fachhochschule Trier den Studiengang *Digitale Medien und Spiele*.

Durch die interdisziplinären und dadurch fachgerechteren Studiengänge ist zu erwarten, dass in den nächsten Jahren in der Spieleentwicklung die Bereiche Game Design und Softwareentwicklung enger zusammenrücken werden. Durch eine umfassendere Sicht auf die Spiele, die sowohl die Perspektive des Game-Designers als auch des Softwareentwicklers berücksichtigt, lassen sich wichtige Kriterien wie der Spielspaß oder implizite, vom Game-Designer nicht dokumentierte Anforderungen (vgl. [11], S. 6 bis 7) in der Anforderungsanalyse beim Softwareentwurf besser erfassen. Der Übergang zwischen Vorproduktions- und Produktionsphase lässt sich dann anhand eines breiteren Wissens und Verständnisses der anderen Arbeitsbereiche von den an der Spieleentwicklung beteiligten Personen besser managen.

Auch der Bereich der Medienerstellung kann von einer engeren Verzahnung mit der Softwareentwicklung profitieren: Die Steigerung des Automatisierungsgrads lässt den Produktionsprozess effizienter gestalten. Der erstellte Spielweltgenerator zeigt, wie Softwaretechnik eingesetzt werden kann, um aufwändige Erstellungsabläufe von Spielwelten zu automatisieren.

Bei zunehmendem Umfang und steigender Komplexität von Spielinhalten und den dadurch zunehmenden Kosten wird die automatisierte Generierung von Inhalten auf Grundlage leistungsfähiger abstrakter Modelle und Architekturen mehr Gewicht bekommen.

Bei der Forschung, die sich mit virtuellen Welten beschäftigt, rücken seit den letzten Jahren immer mehr Ansätze zur automatisierten Generierung in den Fokus, die auf eine präziser kontrollierbare Generierung abzielen (vgl. [14]). Erwähnenswert sind besonders die Ansätze, die Bedienung zur manuellen Erstellung komplexer Szenen auf das grobe Skizzieren von den gewünschten Eigenschaften zu vereinfachen (vgl. [41], S. 354 bis 355). So lassen sich z. B. Berge, Flüsse oder Bereiche mit Städten schnell erstellen. Durch ein konsistentes und semantisches Modell lässt sich die virtuelle Welt dann auf niedrigerem Abstraktionsniveau automatisch generieren. Es ist zu erwarten, dass Techniken zur Modellierung und Generierung realistischer, virtueller Welten zunehmend in der Spieleentwicklung genutzt werden.

Anhang

Bedienschnittstelle der Software

Über die Bedienschnittstelle stehen folgende Tools zur Verfügung, damit die Welt mit Testobjekten gefüllt werden kann:

- Bearbeiten, d. h. Anlegen, Modifizieren und Löschen von Grafik bzw. 3D-Medien wie Texturen, 3D-Modellen, Materialien
- Bearbeiten von Objekteigenschaften, wie z. B. Größe und Informationen, an welchen Positionen das Objekt realistisch vorkommen kann

Objektstruktur der Spielwelt-Daten

Die Klasse *Worldobject* stellt die Oberklasse von Objekten der erzeugten Spielwelt-Daten dar und beinhaltet Schnittstellen zum Rendern, Kollisionserkennung, Dockingmöglichkeit und zur Ein- und Ausgabe von Parametern. Ein vereinfachtes Klassendiagramm ist in Abbildung 5.1 dargestellt.

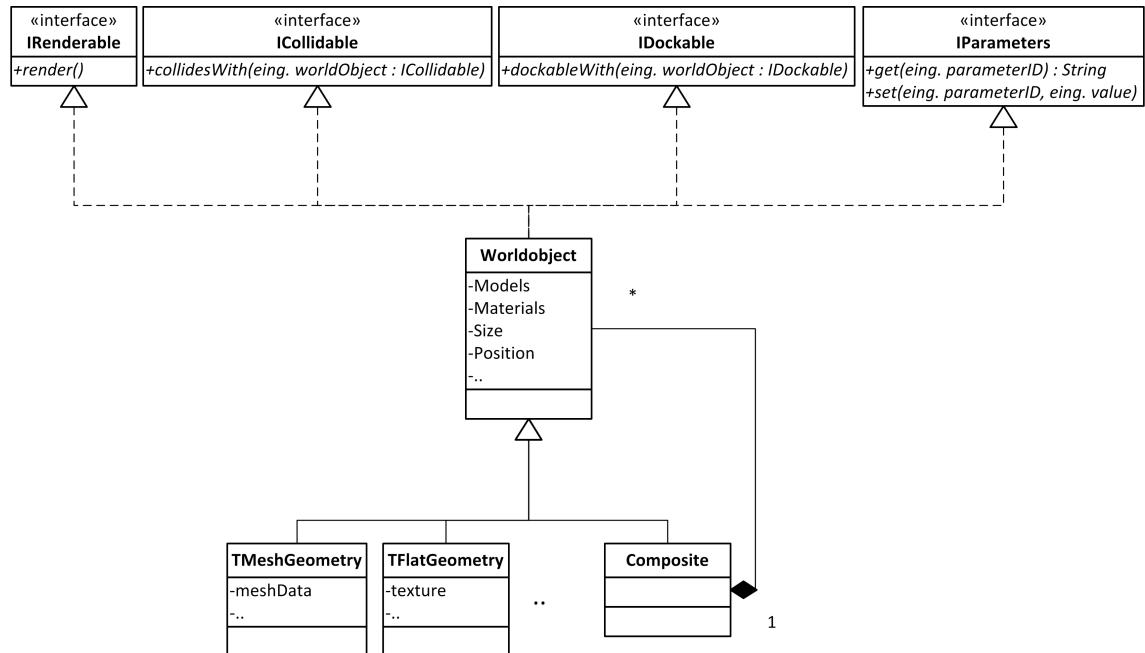


Abbildung 5.1: Vereinfachtes Klassendiagramm

Die Schnittstellen haben die Bedeutung:

Render (IRenderable): Das Objekt wird in die 3D-Szene gezeichnet.

Kollision (ICollidable): Wird ein Objekt in die Welt gesetzt, darf es nicht an eine Stelle gesetzt werden, an der sich bereits ein anderes befindet. Die Schnittstelle liefert die Information, ob bzw. in welchem Umfang ein Objekt mit einem anderen kollidiert.

Docking (IDockable): Die Schnittstelle liefert die Information, ob ein Objekt an einer bestimmten Stelle an ein anderes angedockt werden kann.

Parameter (IParameters): Werte wie z. B. Gewicht, Größe und weitere Eigenschaften werden über diese Schnittstelle gelesen und zugewiesen.

Generierte Landschaften

Die folgenden Bilder der gerenderten 3D-Landschaften wurden erzeugt, indem einige Selectors und Objekte aus der in 3.3.7 beschriebenen Beispielkonfiguration modifiziert wurden.

Bei der letzten Abbildung wurden die Definitionen von zwei Metaebenen modifiziert, sodass die Heightmap wurde mit einer Map von einer Rennstrecke verknüpft wurde. Durch die Modifikation lassen sich innerhalb von Sekunden automatisch beliebige Landschaften mit fertig eingebauter Rennstrecke erzeugen.



Abbildung 5.2: Generierte Insel mit Palmen



Abbildung 5.3: Generierte Wiese



Abbildung 5.4: Generierter See



Abbildung 5.5: Generierte Skyline



Abbildung 5.6: Rennstrecke

Glossar

Artwork

Artwork von Spielen ist ein Sammelbegriff, der die Gesamtheit der grafischen Medien wie z. B. Bilder, Fotos, Zeichnungen und Computergrafiken bezeichnet, die im Zusammenhang einer Spieleentwicklung, z. B. als Skizzen oder zur Präsentation, erstellt werden.

Balancing

Als Balancing¹ von Spielen wird die Ausgeglichenheit von Taktiken und Strategien im Spielverlauf bezeichnet. Eine solche, in der Regel vom Game-Designer nicht beabsichtigte Unausgeglichenheit kann zur Folge haben, dass sich bei steigender Komplexität während des Spielverlaufs bestimmte Strategien als überlegen herausstellen und somit eine Unfairness zwischen Spielern, die unterschiedliche Strategien verfolgen, entsteht bzw. aufgrund der Wahl einer bestimmten Strategie das Spiel zu einfach oder schwierig für den Spieler wird.

Ego-Shooter

Mit Ego-Shooter² wird die Spielegattung bezeichnet, bei der sich der Spieler in einer 3D-Spielwelt bewegt, mit Schusswaffen seine Gegner bekämpft und die Egoperspektive verwendet, d. h. die Kameraperspektive aus Sicht der Spielfigur verwendet wird.

Genre

Als Genre von Spielen oder Spielgenre werden verschiedene Gattungen von Spielen hauptsächlich nach Art der Interaktion und der Spielmechanik unterschieden. Bekannte Spielgenres sind Adventures, Ego-Shooter, Rollenspiele, Strategiespiele und Simulationsspiele.

¹auch Balance

²auch First-Person-Shooter

Heightmap

Heightmaps³ sind 2D-Datenfelder, die ein Höhenfeld in Form eines Skalarfelds beschreiben. Jedem Ort ist ein Wert zugeordnet, der eine Position im 3D-Raum beschreibt. Dadurch können die Höheneigenschaften von Landschaften beschrieben und gespeichert oder visualisiert werden.

Level

Die Gliederung von Spielen in Levels ist mit der von Büchern in Kapitel vergleichbar. Ein Level stellt einen Spielabschnitt dar, den der Spieler erst bewältigen muss, bevor er in den nächsten Level gelangt. Leveledesign bezeichnet das Erstellen von Levels.

Middleware

Middleware bezeichnet Software, die als innere Ebene eines Softwaresystems mit mehreren Ebenen zwischen oberer und unterer Ebene vermittelt, indem sie Schnittstellen und Dienste bereitstellt. In der Spieleentwicklung werden auch allgemein Subsysteme für Teilbereiche als Middleware bezeichnet. Diese werden oft von Anbietern entwickelt und angeboten, die sich auf einen Bereich wie Grafik, Physik oder Künstliche Intelligenz spezialisiert haben, und werden von Spieleentwicklern in die Spielsoftware eingebunden.

Rendern

In der Computergrafik bezeichnet das Rendern⁴ die Erzeugung eines Bildes aus Rohdaten. Bei Computerspielen mit 3D-Grafik bestehen die Rohdaten aus räumlichen Daten, die u. a. 3D-Knoten, Materialeigenschaften, Lichtquellen beinhalten. Das Bild wird dabei aus der durch technische Parameter definierten Perspektive des Betrachters erzeugt.

Rollenspiele

Das Spielgenre Rollenspiel (auch *RPG*⁵) bezeichnet Spiele in einer fiktiven Spielwelt mit einer komplexen Handlung. Der Spieler taucht in die Rolle seiner Spielfigur und kann durch eigene Entscheidungen den Handlungsverlauf beeinflussen. Oft in Rollenspielen verwendete Spielemente sind *Queste*, das sind Aufgaben, die der Spieler erledigen muss, und das *Leveling*: Durch gesammelte Fortschrittpunkte kann der Spieler spezielle Fähigkeiten seiner Figur wie z. B. Schnelligkeit steigern, oder

³auch *Heightfields* genannt

⁴auch *Rendering* und *Bildsynthese*

⁵engl. *Role Play Game*

bestimmte Fähigkeiten erlangen. Dies erlaubt eine individuelle Spezialisierung und damit Strategiewahl.

Story

Wie bei Filmen bezeichnet die Story bei Spielen die Handlungsabfolge. Je nach Spielgenre und Art des Einsatzes kann eine Story nur als Rahmenhandlung und Illustration verschiedener Spielstufen fungieren oder in direktem Zusammenhang mit Spielgeschehen und dem Spieler interagieren. Ein Unterschied zu Filmproduktionen ist die Möglichkeit mehrerer und unterschiedlich ausgeprägter Handlungsstränge, die sich aus dem Spielgeschehen bzw. durch Entscheidungen des Spielers ergeben.

Literaturverzeichnis

- [1] E. Aarseth, Leveling the playing field in *Playing Research: Methodological approaches to game analysis, Digital Arts and Culture conference*, Spilforskning.dk, Melbourne, 2003.
- [2] E. Adams, „Inside The Fun Factory, Stage 3: Production” in *Break into the Game Industry: How to Get A Job Making VideoGames*, McGraw-Hill/Osborne, 2003.
- [3] J. Bacher; A. Pöge; K. Wenzig, *Clusteranalyse. Anwendungsorientierte Einführung in Klassifikationsverfahren*, Oldenbourg, München, 2010.
- [4] R. Bartle, „Hearts, clubs, diamonds, spades: Players who suit MUDs” in *Journal of MUD Research vol. 1 (1)*, 1996, <http://mud.co.uk/richard/hcds.htm> (01.04.2012, 6:24).
- [5] E. Bethke, „Introduction to Game Development” in *Game development and production*, Wordware Publishing, Inc., 2003.
- [6] L. Bishop et al., „Designing a PC game engine” in *IEEE Computer Graphics and Applications, Volume 18 Issue 1*, 1998.
- [7] S. Björk; J. Holopainen, *Patterns in Game Design*, Game Development Series, Charles River Media, 2005.
- [8] B.W. Boehm, „Guidelines for Verifying and Validating Software Requirements and Design Specification” in *IEEE Software*, 1984.
- [9] B.W. Boehm, „A Spiral Model of Software Development and Enhancement” in *IEEE Computer 21*, 1988.
- [10] J. Bragge; J. Storgårds, „Profiling Academic Research on Digital Games Using Text Mining Tools” in *Situated Play, Digital Games Research Association (DiGRA) 2007 Conference*, Tokyo, 2007.
- [11] D. Callele et al., „Requirements Engineering and the Creative Process in the Video Game Industry” in *Proceedings of the 13th IEEE International Conference on Requirements Engineering (RE'05)*, 2005.
- [12] D. Callele; E. Neufeld; K. Schneider, „Emotional Requirements in Video Games” in *14th IEEE International Requirements Engineering Conference (RE'06)*, 2006.

- [13] S. Che et al., „A Performance Study of General Purpose Applications on Graphics Processors using CUDA” in *Journal of Parallel and Distributed Computing*, Elsevier, Amsterdam, 2008.
- [14] J. Doran; I. Parberry, „Controlled Procedural Terrain Generation Using Software Agents” in *IEEE Transactions on Computational Intelligence and AI in Games*, 2010.
- [15] D. Dumont, *Spielspaß kalkulieren* in Making Games Magazin, 2012, Ausgabe 01.
- [16] A. Fall; J. Fall, „A domain - specific language for models of landscape dynamics” in *Ecological Modelling Vol. 141*, Elsevier, Amsterdam, 2001.
- [17] J. Flynt; O. Salem, „Philosophy of Software Engineering and Game Development” in *Software Engineering for Game Developers*, Thomson Course Technology PTR, Boston, 2005.
- [18] T. Fullerton et al., *Game Design Workshop: Designing, Prototyping, and Playtesting Games*, Elsevier/Morgan Kaufmann, Amsterdam, 2004.
- [19] T. Fullerton et al., *Game Design Workshop: A Playcentric Approach to Creating Innovative Games*, Elsevier/Morgan Kaufmann, Amsterdam, 2008.
- [20] E. Gamma et al., *Design Patterns: Elements of Reusable Object Oriented Software*, Addison-Wesley, Professional Computing Series, 1995.
- [21] S. Greuter; N. Stewart; G. Leach, „Beyond the Horizon - Computer generated, three-dimensional, infinite virtual worlds without repetition in real-time” in *Image Text and Sound Conference*, 2004.
- [22] W. Hesse et al., „Ein Begriffssystem für die Softwaretechnik” in *Informatik-Spektrum 7.4*, 1984.
- [23] J. Holopainen; J. Kuittinen, „Some Notes on the Nature of Game Design” in „*Breaking New Ground: Innovation in Games, Play, Practice and Theory*”, Digital Games Research Association (DiGRA) 2009 Conference, London, 2009.
- [24] P. V. C. Hough, *Method and means for recognizing complex patterns*, U.S. Patent 3 069 654, 1962.
- [25] D. Kafura; R. R. Reddy, „The Use of Software Complexity Metrics in Software Maintenance” in *IEEE Transactions on Software Engineering Vol. SE-13, No. 3*, 1987.
- [26] J. Keller, „Ausbildung und Arbeiten in der Computer- und Videospiel-Industrie” in *Die Gamesbranche - Ein ernstzunehmender Wachstumsmarkt*, 2. aktualisierte Auflage, Hessisches Ministerium für Wirtschaft, Verkehr und Landesentwicklung, Wiesbaden, 2010.
- [27] R. Klein, „Voronoi-Diagramme” in *Algorithmische Geometrie: Grundlagen, Methoden, Anwendungen*, 2. Auflage, Springer Verlag, Berlin, 2005.

- [28] C. Klimmt, *Der Stand der Wissenschaft: Spielspaß erklären* in Making Games Magazin, 2012, Ausgabe 01.
- [29] B. Kreimeier, *The Case For Game Design Patterns*, 2002, http://www.gamasutra.com/features/20020313/kreimeier_3.htm (02.04.2012, 2:06).
- [30] B. Meyer, *Object Oriented Software Construction*, Prentice Hall, 1988.
- [31] J. Olsen, *Realtime Procedural Terrain Generation*, Technical Report, University of Southern Denmark, 2004.
- [32] PricewaterhouseCoopers, „Executive Summary“ in *Global Entertainment and Media Outlook: 2008-2012*, 2008.
- [33] P. Rechenberg, *Was ist Informatik?: Eine allgemeinverständliche Einführung*, 2. Auflage, Carl Hanser Fachbuch Verlag, München, 1993.
- [34] A. Rollings; D. Morris, „Roles and Divisions“ in *Game Architecture and Design: A New Edition*, New Riders Publishing, 2004.
- [35] R. Rouse III, *Game Design: Theory and Practice*, 2. Ausgabe, Wordware Game Developer's Library, Jones and Bartlett Publishers, Sudbury Massachusetts, 2005.
- [36] J. Schell, *The Art of Game Design: A book of lenses*, Elsevier/Morgan Kaufmann, Amsterdam, 2008.
- [37] C. Schmidt, *Weckruf für Emotionen: Spielspaß erfühlen* in Making Games Magazin, 2012, Ausgabe 01.
- [38] R. M. Smelik et al., „A Proposal for a Procedural Terrain Modelling Framework“ in *Proceedings of Eurographics Virtual Environments symposium (EG-VE)*, 2008.
- [39] R. M. Smelik et al., „A Survey of Procedural Methods for Terrain Modelling“ in *Proceedings of CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS)*, Amsterdam, 2009.
- [40] R. M. Smelik et al., „Integrating procedural generation and manual editing of virtual worlds“ in *Proceedings of the ACM Foundations of Digital Games*, ACM Press, 2010.
- [41] R. M. Smelik et al., „A declarative approach to procedural modeling of virtual worlds“ in *Computers & Graphics, Volume 35, No. 2*, Elsevier, Amsterdam, 2011.
- [42] „Software engineering“ in *IEEE Standard Glossary of Software Engineering Terminology*, 1990.
- [43] I. Sommerville, „Einführung“ in *Software Engineering*, 8. Auflage, Pearson Studium, 2007.

- [44] C. D. Tomlin, „Cartographic Modelling” in D. J. Maguire; M. F. Goodchild; D. W. Rhind (eds.) *Geographical Information Systems: Principles and Applications*, John Wiley & Sons, 1991.