Techniken der Prozeduralen Generierung für Dreidimensionale Dungeons in Videospielen

Bachelor-Thesis

zur Erlangung des akademischen Grades B.Sc.

Börje Santen

2058479

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Design, Medien und Information
Department Medientechnik

Erstprüfer: Prof. Dr. Edmund Weitz

Zweitprüfer: Prof. Dr. Andreas Plaß

Hamburg, 08. 05. 2015

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelor-Thesis mit dem Titel:

„Techniken der Prozeduralen Generierung für Dreidimensionale Dungeons in Videospielen“

selbständig und nur mit den angegebenen Hilfsmitteln verfasst habe. Alle Passagen, die ich wörtlich aus der Literatur oder aus anderen Quellen wie z. B. Internetseiten übernommen habe, habe ich deutlich als Zitat mit Angabe der Quelle kenntlich gemacht.

(Unterschrift)

Abstract

My first contact with procedural content was the game Diablo (1997) and his successor Diablo 2 (2000), they generated the world, object properties and names of the enemies.
I was especially fascinated from the random looking dungeons, this is why I always got back to similar games. The application of procedural generation (PG), independent from a specific technique or area of application, is a important part of video game production. The necessity of this technique is increasing over the years in all operational areas for the development.
The use of PG in an industry in which I'm interested, for an application which is fascinating me for over 15 year has motivated me to write my bachelorthesis over “Technique of Procedural Generation for three-dimensional Dungeons in Videogames”.

Inhaltsverzeichnis

1 Vorwort

1.1 Motivation

Meine erste Begegnung mit prozeduralen Inhalten war das Spiel Diablo (1997) und seine Fortsetzung Diablo 2 (2000), sie generierten die Welt, Objekteigenschaften und Namen der Gegner. Besonders die mir zufällig wirkenden Dungeons hatten es mir angetan und ließen mich über die Jahre immer wieder zu ähnlichen Spielen zurück kommen.
Die Anwendung von Prozeduraler Generierung (PG), ganz unabhängig von Technik und Anwendungsgebiet, ist ein fester Bestandteil der Videospiel Produktion. Ihre Notwendigkeit in der Entwicklung steigt seit Jahren stetig in allen ihren möglichen Einsatzbereichen an.
Die PG in einer Industrie die mich interessiert, für eine Anwendung die mich seit über 15 Jahre fasziniert, hat mich dazu bewogen diese Bachelorarbeit zu schreiben.

1.2 Ziel der Arbeit

In meiner Bachelorarbeit möchte ich die Techniken der Prozeduralen Generierung von Dreidimensionalen (3D) Dungeons beschreiben. Sie sollen in Theorie und Praxis erklärt werden, damit auch für Materie-fremde ihre Funktion und Anwendung klar verständlich und ersichtlich wird. Da es, auf Grund der reinen Anzahl, nicht möglich ist alle möglichen Techniken in dieser Tiefe zu beschreiben, wird sich hier auf etablierte Beispiele beschränkt. Welche sind Binäre Raumpartitionierung, Zellulare Automaten und Generative Grammatik.
Als praktischen Abschnitt der Arbeit wird ein rudimentärer 3D Dungeon Generator entwickelt, dieser soll mit der Hilfe von Generativer Grammatik in der Unity3D-Engine erstellt werden. Er wird unter rein technischen Gesichtspunkten erstellt und lässt die variablen Ansprüche aus dem Gamedesign außen vor.
Die Anforderungen, die erfüllen werden müssen:

- Start- und Endpunkt werden gesetzt und sie sind zueinander erreichbar
- Start- und Endpunkt besitzen einen Mindestabstand
- Keine Löcher in der Geometrie, fehlender Boden, Wände
- jede Räumlichkeit muss erreichbar sein, keine isolierten Bereiche
- Raumstrukturen über mehrere Ebenen

1.3 Was ist ein Dungeon

Eine Dungeon oder in der Deutschen Sprache das Verlies wird mit einem mittelalterlicher Kerker assoziiert, was hier nicht zutrifft. Im Videospielkontext hat es eine weitaus breitere Bedeutung, die ich an dieser Stelle für die Bachelorarbeit spezifizieren möchte. Es ist ein Szenario unabhängiges Netzwerk oder eine Aneinanderreihung von Fluren und oder Räumen mit variabler Anzahl von Start- und Endpunkten. Sie können mehrere Ebenen besitzen und beim Durchqueren stellen sich Hindernisse wie Gegner, Rätsel oder verschlossene Durchgänge dem Spieler in den Weg.
Als dreidimensional gilt ein Dungeon, wenn es über drei Achsen visualisiert wird.
Das schließt einen zweidimensionalen Raumplan mit hochgezogenen Wänden bis hin zu einem Höhlensystem über mehrere Stockwerke ein.
Von der Bezeichnung Level distanziere ich mich bewusst in der Beschreibung vom Dungeon, auch wenn sie oft gleichgestellt werden. Der Grund ist, dass der Begriff Dungeon in diesem Kontext länger verwendet wird und dass das Wort Level mit der Ebene oder Stockwerk gleichgesetzt werden kann.

2 Prozedurale Generierung

2.1 Was ist Prozedurale Generierung

Es bezeichnet Methoden für die automatische Erstellung von digitalen Inhalten und sie werden in vielen Industrien eingesetzt. Es beginnt bei den Visuellen Effekten im Filmen und geht über den gesamten Anwendungsbereich in Videospielen bis hin zu klassischen, künstlerischen Inhalten wie Musik und Malerei.
Ein oft bestehendes Missverständnis ist, dass es sich nicht um eine Zufallsgenerierung handelt, sondern um determinierte Algorithmen. Ein Algorithmus heißt determiniert, wenn er mit gleichen Startbedingungen und Parametern stets das gleich Ergebnis liefert. Dies ist wichtig, damit bei gleichem Input immer derselbe Output erreicht wird und so auch gezielt Inhalte generiert werden können.

2.2 Einsatz in Videospielen

Prozedurale Techniken können in allen Bereichen eines Videospieles angewendet werden. Dazu zählen Texturen, Terrain, Vegetation, Straßen, Städte, Dungeons, Galaxien, Missionen, Objekte, Sound und viele mehr. Zu Beginn wurden sie nur in einzelnen Gebieten verwendet und bildeten das Hauptmerkmal des Spieles. Im Verlauf der Zeit stieg die Anzahl der Anwendungsgebiete für die PG in Spielen eingesetzten wurde. Dies ging soweit, dass ganze Produkte veröffentlicht wurden die komplett Prozedural Generiert waren.

2.2.1 Ihre erste Verwendung

Zu den ersten Titeln die prozedurale Techniken kommerziell anwendeten, gehörten Rogue (1980), Rescue On Fractalus (1984) von Lucasfilm Games und Elite (1984). Rogue erstellte bei jedem Start einen neuen Dungeon, der mit ASCII-Zeichen visualisiert wurd.
Rescue On Fractalus verwendete Fraktale, daher auch der Name des Spieles,
um eine bergige Planetenoberfläche darzustellen. Elite ging schon weiter und generierte eine ganze Galaxie mit Sonnensystemen, Planeten und Missionen prozedural.
Die automatisch generierten Inhalte in diesen Spielen, waren auch aus spielerischer Sicht ihr Hauptmechaniken. Alle diese Spiele wurde von kleinen Personengruppen entwickelt und durch den Mangel an Ressourcen wie Speicherplatz hätten sie nie ohne PG solche großen Welten erschaffen können. Auch bildete die grundsätzliche Benutzung von solchen Techniken damals eine Ausnahme und ließen sie so aus der Masse heraus stechen.

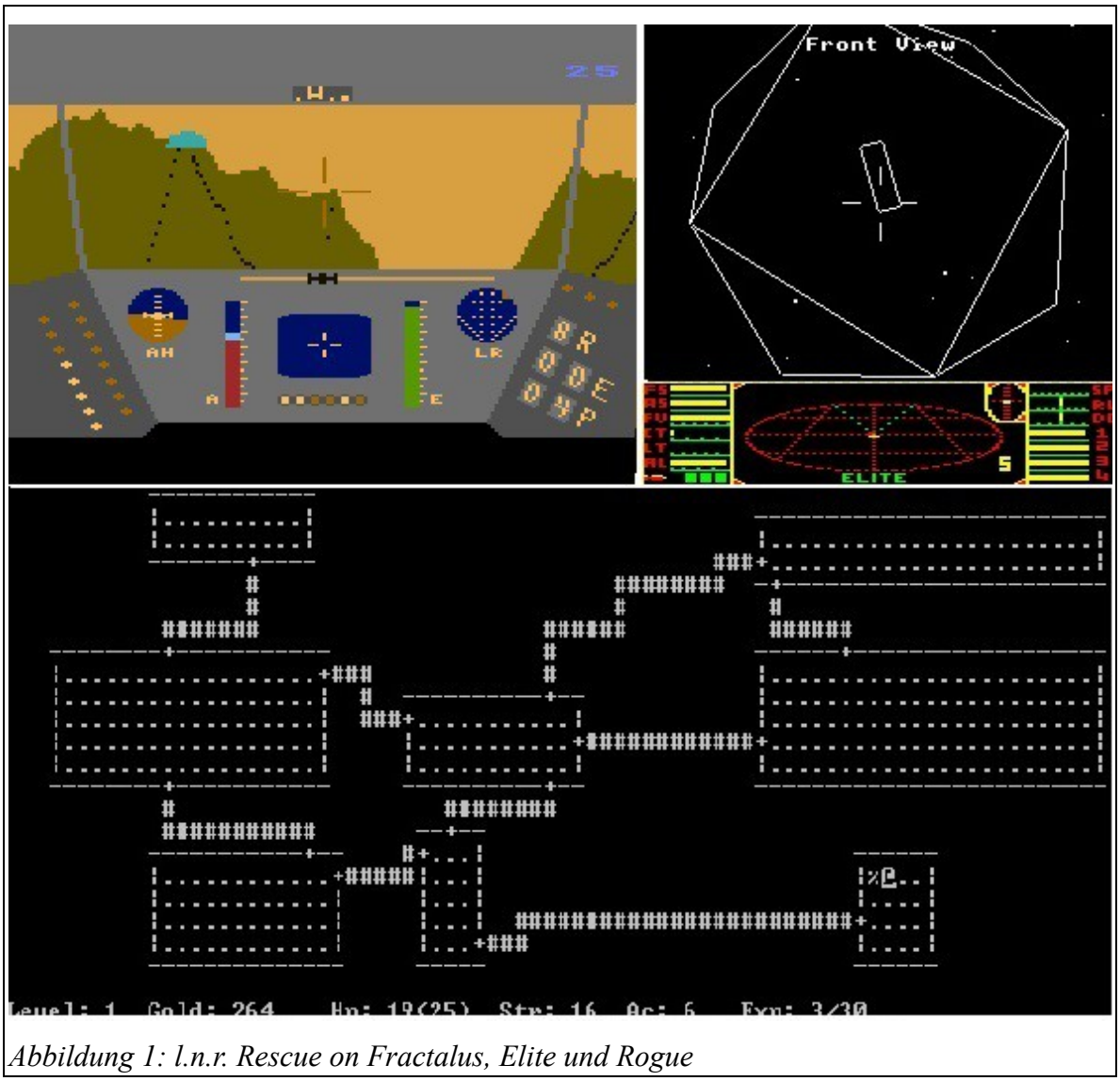

Abbildung 1: l.n.r. Rescue on Fractalus, Elite und Rogue

2.2.2 Heutige Verbreitung

Heute ist die Nutzung von PG Techniken in der Videospielindustrie in allen Bereichen etabliert, sie lassen sich aber dennoch in zwei Strömungen unterteilen. Zuerst ihre Verwendung als Produktionswerkzeug um sehr viel Inhalte in einem angemessenen Zeitrahmen zu erstellen, was vor allem für große Produktionen genutzt wird. Die zweite Strömung ist die Nutzung von Prozeduralen Techniken als Spielmechanik, ein Ansatz der gerne von kleinen Entwicklern genutzt wird um ein herausstreichendes Merkmal zu haben. Ein Vorteil den beide genießen ist, dass das automatische Generieren von Inhalten Zeit und somit auch Geld spart, etwas was bei Entwicklern jeder Größe beliebt ist.
Mit am weitesten verbreitet ist das Prozedurale Generieren von Texturen, um eine Vielzahl von unterschiedlichen Mustern für Bäume, Wände oder Böden zu haben. Diese müssen maximal noch einmal angepasst werden und können direkt verwendet werden.
Auch die Flora in Videospielen ist ein Gebiet, wo PG häufig zum Einsatz kommt.
Für die Erstellung von Vegetation und ihre großflächige Verpflanzung in Virtuellen Welten ist eine Software Namens „SpeedTree“ weit verbreitet. Mit ihr lassen sich alle Teile der Pflanzenwelt in vielen Variationen erschaffen und einzeln platzieren oder für ein markiertes Gebiet eine Vegetation mit sich nicht wiederholender Flora generieren.
Etwas von dem in einigen Spielen sehr viel benötigt wird, sind Ausrüstungsobjekte für den Spielcharakter. Das betrifft typischerweise Objekte wie Waffen, Rüstungen und Verbrauchsgegenstände, die sich hauptsächlich durch ihre verschiedenen Eigenschaftswerte unterscheiden. Das Spiel Borderlands (2009) machte dies zu einem Verkaufspunkt und schrieb in ihrer Werbung das sie 17 Millionen verschiedene Waffen in ihrem Spiel haben, natürlich erstellt durch Prozedurale Generierung.
Das prozedurale Erstellen von Terrain wir am häufigsten für isolierte Landmassen wie Inseln verwendet, ist aber auch für Landarten einsetzbar. Es werden gezielt bestimmte Formen generiert und dann mit der Hand nachgearbeitet. Automatische Terrain Erstellung ohne menschliche Nachbearbeitung ist eher in abstrakten Spielen zu finden, wie zum Beispiele Minecraft (2009).
Die automatische Produktion von Aufgaben in Spielen übernehmen sogenannten Mission Generatoren, sie sind in der Lage eine große Anzahl von Nebentätigkeiten zu produzieren. Sie sollen große Spielwelten mit Leben erfüllen, damit der Spieler nie ohne eine Aufgabe ist und länger an das Spiel gebunden wird.

Die Erzeugung einer Weltraumumgebung wie einer Galaxie mit kompletten Sonnensystemen und Asteroidenfeldern wird in Elite: Dangerous (2014), die aktuellste Fortsetzung von Elite, und EVE Online (2003) umgesetzt. Besonders die Spielreihe Elite ist dafür bekannt geworden mit Zufallszahlengeneratoren Galaxien zu erzeugen, wobei jeder Asteroid oder Planet bis hin zur kleinsten Spezifikation automatisch erstellt wird.

2.2.3 Der Einsatz speziell für Dungeon

Das Generieren von Dungeons gehörte mit Rogue zu den ersten Anwendungsgebieten von prozeduralen Techniken in Spielen. Aktuell haben diese eine besonders große Beliebtheit bei kleinen oder Nischen Entwicklern vor allem für 2D Spiele. Dabei setzen sie PG nicht nur wegen der Zeitersparnis an, sondern als bewusste Design Entscheidung und Spielmechanik. Beschränkungen auf einzelne Spiele-Genre sind dort nicht zu finden aber eine starke Anlehnung an Spiele aus den Achtzigern.
Bei großen Entwicklern, die ein Millionenpublikum ansprechen wollen, sind prozedurale Dungeon kaum zu finden. Wenn sind sie zu meist spielerische Klone von der Spielreihe Diablo. Die Ausnahme bilden die beiden Titel Warframe (2013) und Bloodborne (2015), welche prozedurale Techniken für Dungeons verwendet. Warframe ist ein Action orientiertes Spiel, das für jede Mission einen 3D Dungeon mit verschiedenen Ebenen erzeugt, welcher sich aus fertigen Elementen zusammensetzt. Bei Bloodborne ist es möglich die generierten Dungeon zu speichern und zu teilen, damit sich Spieler an derselben automatisch generierten Herausforderung messen können.

2.2.4 Zukünftige Anwendungen und Projekte

In naheliegender Zukunft soll ein Spiel, das alle Anwendungen in sich verbinden möchte, namens „No Man's Sky“ veröffentlicht werden (geplant noch 2015). Es soll eine ganze Galaxie mit Sonnensystemen und begehbaren Planeten, welche alle angereist werden können, geben. Dort wird auf dem Planeten die gesamte Geographie, Fauna und Flora prozedural generiert. Wenn es diese Versprechen erfüllen kann, könnte es den großflächigen Einsatz von prozeduralen Techniken voranbringen.
Bisherige Erfahrungen sprechen gegen ihren Erfolg. Das erste Spiel das versuchte eine so große und komplexe Welt auf diese Weise zu erstellen ist es nicht. Schon 1996 wurde das Spiel The Elder Scrolls 2: Daggerfall veröffentlicht und hatte nach eigenen Angaben die doppelte Größe von Großbritannien mit über 5000 Dörfern und hunderttausenden Nichtspieler Charaktere. Bekannt wurde es dann mehr für seine große Anzahl von Fehlern.

2.3 Einsatz in anderen Medien

Abseits von Videospielen findet PG auch ihre Anwendung in Filmen, besonders in Animationsfilmen liegt deren Nutzung nahe. Für die Firma Pixar, die mit ihren Werken Weltbekanntheit erreicht hat, ist PG ein essenzieller Bestandteil ihres Arbeitsprozesses.
Die Inhalte ihrer Filme werden weniger von Menschen mit einer künstlerischen Ausbildung erstellt, sondern von Softwareentwicklern aus der Informatik, Mathematik oder Physik.
Auch die hauseigene Software, mit den Namen RenderMan, wird mit jedem ihrer Projekte mit weiteren Anwendungen für PG ausgebaut und somit auch immer mächtiger.
Außerhalb von Pixar ist einer der bekanntesten Filme der auf PG setzt der Film Avatar (2009). Dort wurde es für eine abwechslungsreiche Vegetation bis hin zu den herumfliegenden Insekten eingesetzt. In der filmischen Fortsetzung, die 2017 erwartet wird, kann man von einer qualitativen und quantitativen Steigerung der Nutzung von prozeduralen Techniken ausgehen.

2.4 Stand der Technik und Anwendung

Die Techniken die für die PG der Inhalte verantwortlich sind, haben sich über die Zeit nicht verändert. Gut zu sehen ist das an der Spielreihe Elite (1984), welche mit Elite: Dangerous (2014) dieselben Techniken einsetzt. Mit der zunehmenden Rechenleistung ist anzunehmen, dass Techniken die zuvor auf Grund ihrer Laufzeit nicht sinnvoll waren nun nutzbar sind oder man neue Möglichkeiten gefunden hat sie einzusetzen.

3 Techniken der Prozeduralen Generierung

Techniken, die es ermöglichen automatisch Inhalte zu generieren, gibt es viele.
Es werden weniger wenn man wie hier ihre Anwendung spezifiziert und sich die Etablierten von ihnen heraussucht. Bei ihnen handelt es sich um Binäre Raumpartitionierung, Zellulare Automaten und Generative Grammatik. Um zu zeigen wie die vorgestellten Techniken funktionieren, werden sie hier in Theorie und in ihrer praktischen Anwendung beschrieben. Es wird gezeigt, dass sie sich den Vorgaben unterwerfen und damit als Techniken sinnvoll eingesetzt werden können.
Die Vorgaben die diese Algorithmen erfüllen müssen sind:

- Sie müssen determiniert sein, damit ihre Ergebnisse reproduzierbar sind
- Das Ergebnis soll eine durchgängige Geometrie erzeugen, keine isolierten Bereiche
- Sinnvolles automatisches Setzen von Start und Endpunkt
- Die Algorithmen müssen eine geringe Laufzeit haben

3.1 Binäre Raumpartitionierung

3.1.1 Ursprung

Binäre Raumpartitionierung, öfter unter der englischen Bezeichnung Binary Space Partitioning (BSP) zu finden, ist eine Technik die zu aller erst von Robert A. Schumacher im Jahre 1969 beschrieben wurde. Sein Ziel war es, damit das Sortieren von Polygonen zu beschleunigen. Die Erweiterung des Modells in die dritte Dimension fand 1980 von Henry Fuchs statt. Daraus ist auch das größte Anwendungsgebiet dieser Technik entstanden die Computergrafik. Dort wird sie für die beschleunigte Kollisionserkennung und Verdeckungsberechnung von Polygonen verwendete. Große Verbreitung erlangte die Technik durch ihre Einsatz in einem der ersten 3D-Shooter Namens Doom (1993).
Das darauf aufbauende Konzept lässt sich auch gut für PG einsetzen, wenn man einen 2D oder 3D Raum in zufällige Räume aufteilen möchte.

3.1.2 Definition

Bei binärer Raumpartitionierung geht es darum einen Raum aufzuteilen um daraus eine hierarchische Struktur zu bilden, die durch einen Binärbaum dargestellt werden kann.
Dies geschieht durch das Aufteilen von n-dimensionalen Daten durch eine Menge von Hyperebenen. Im eindimensionalen Raum ist eine Hyperebene ein Punkt,
im zweidimensionalen eine Gerade und für den dreidimensionalen Raum ist es die Ebene.

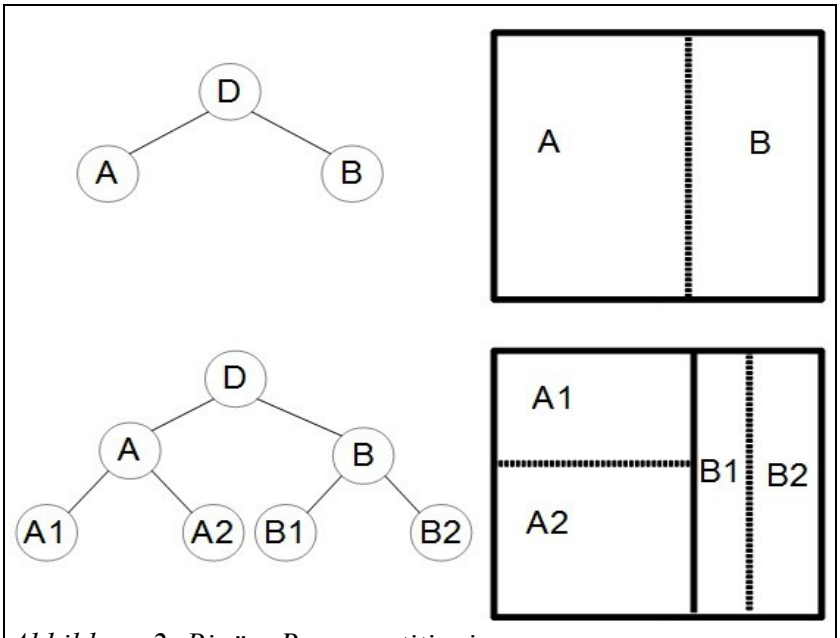

Abbildung 2: Binäre Raumpartitionierung

In Abb. 2 ist zu sehen wie dies für einen zweidimensionalen Raum aussehen könnte. Rechts ist der Raum zu sehen der durch die Hyperebenen, gezeichnet durch die gestrichelten Linien, getrennt ist. Auf der linken Seite ist zu beobachten, wie sich der Binärbaum bildet und mit jeder Hyperebene zwei neue Blätter bekommt.

3.1.3 Anwendung

Die Anwendungen sollen die fertige Partitionierung eines Raumes zeigen und den dazugehörigen Binärbaum. Anhand davon soll dann gezeigt werden wie man damit einen Dungoen generieren kann.

3.1.3.1 Fertige Raumpartitionierung

In Abb. 2 sieht man den Beginn der Raumpartitionierung und bei Weiterführung dieses Prozesses entsteht ein Bild wie in Abb. 3, der dazugehörige Binärbaum ist in Abb. 4 zu sehen. Wichtig ist das sich ein vollständig balancierter Binärbaum gebildet hat, etwas was spezifisch für dieses Beispiel ist und nicht allgemein gültig. Dies ist sehr gut in Abb. 2 zu betrachten, nachdem der Raum in A und B unterteilt wurde und im Folgeschritt jeweils weiter in A1, A2, B1 und B2 aufgeteilt wurden. Der Prozess wurde noch weitere zweimal auf alle Blätter des Baumes angewendet, wodurch der aufgeteilte Raum aus Abb. 3 entstand. Natürlich kann die Partitionierung unendlich fortgeführt werden, was aber für unsere spezifische Anwendung nicht nötig ist. Wenn der Raum fein genug zerlegt wurde, ist die binäre Raumpartitionierung abgeschlossen und es kann mit dem Ergebnis weiter gearbeitet werden.

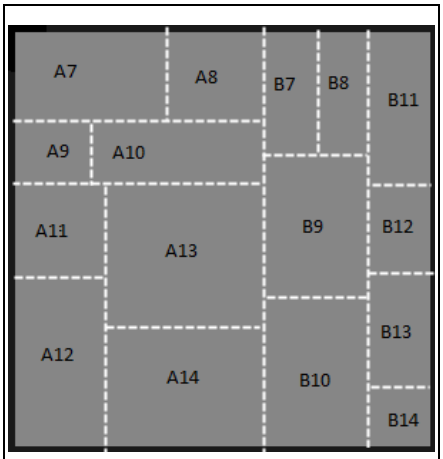

Abbildung 3: Fertig aufgeteilter Raum

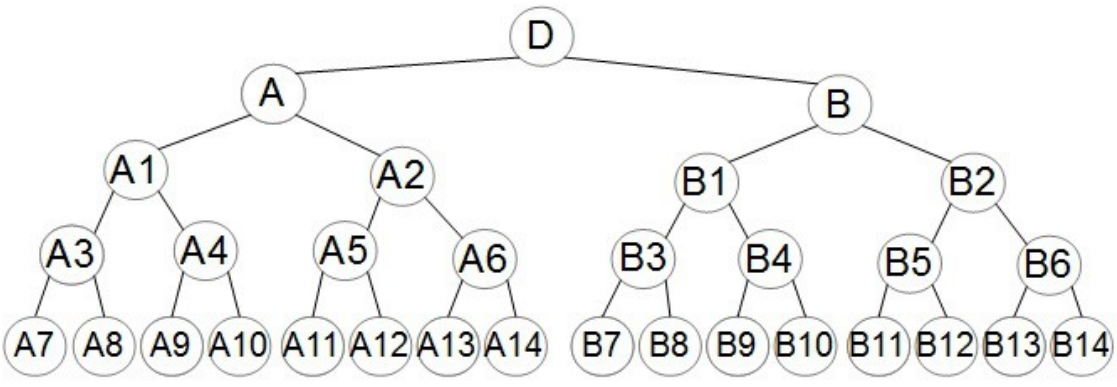

Abbildung 4: Binärbaum zu Abb. 3

3.1.3.2 Verwendung für einen Dungeon

Mit dem fertig partitionierten Raum aus Abb. 3 ist die Hälfte der Arbeit für die Generierung eines Dungeon abgeschlossen. Um aus den Teilräumen nun Räume für einen Dungeon zu machen, werden sie mit Rechtecken in variabler Größe gefüllt. Dabei ist nur zu beachten, dass sie nicht größer werden, als ihr Teilraum zusehen in Abb. 5. Nun ist zwar ein zweidimensionaler Raumplan vorhanden, die Einzelgebiete sind aber noch getrennt.
Für die Erzeugung von Korridoren greifen wir wieder auf den Binärbaum aus der Abb. 4 zurück. Beim Betrachten des Baumes kann man erkennen, dass es sich anbietet wenn die Blätter der inneren Knoten verbunden werden. Von unten angefangen würde man das die Teilräume A7 und A8 bzw. B7 und B8 usw. verbinden. Der Korridor wird durch einen schwarzen Balken dargestellt und bei seiner Platzierung muss überprüft werden, ob er beide Rechtecke der Teilräume auch berührt. Wenn das für alle Blätterpaare der untersten Ebene geschehen ist, entsteht ein Aufbau wie in Abb. 6 zusehen ist.

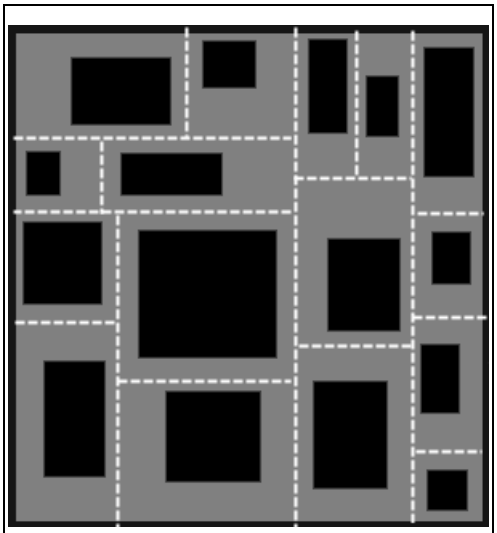

Abbildung 5: Partitionierter Raum gefühlt mit Rechtecken

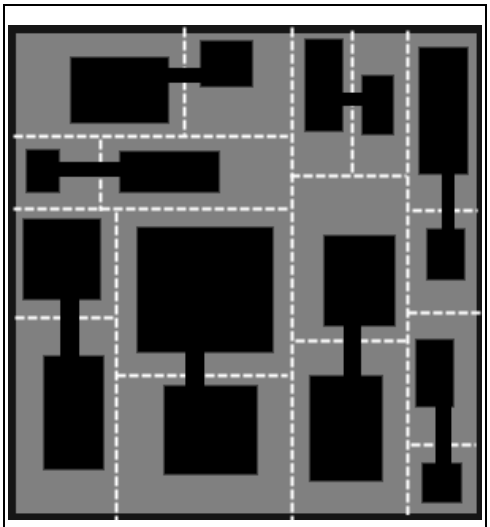

Abbildung 6: Verbindung der untersten Blätter über ihre inneren Knoten

Es folgt eine Wiederholung des Prozesses für eine Ebene höher im Binärbaum, siehe Abb. 7, und es wird langsam eine nutzbare Struktur sichtbar. Nachdem der gesamte Baum durchgegangen ist, erhält man das Bild aus Abb. 8 und es ist ein möglicher Dungeon entstanden.

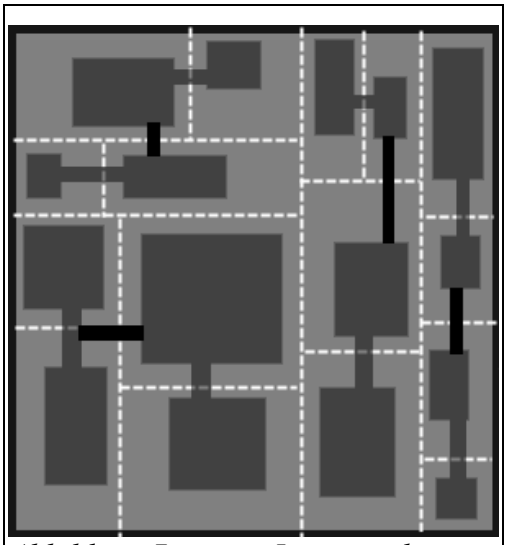

Abbildung 7: zweite Iteration der Korridor Platzierung

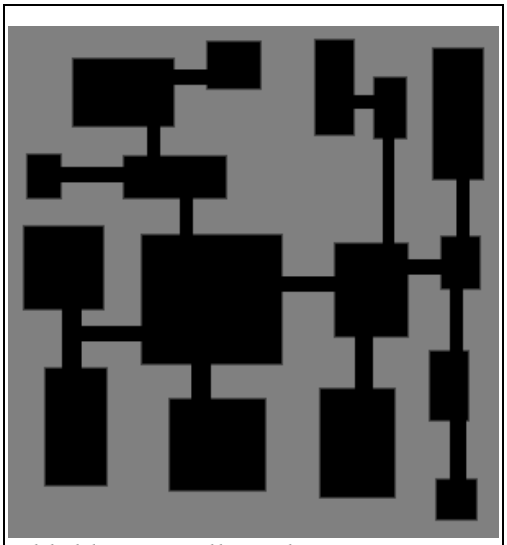

Abbildung 8: Alle Teilräumen zu einem Verbunden

Auf die hier geschilderte Weise wurden die Dungeons für das Spiel Rogue erstellt und sie wird in derselben oder einer modifizierten Weise noch heute angewendet. Beim Betrachten des endgültigen Ergebnisses werden die Vorteile die es noch heute relevant machen ersichtlich. Es ist eine durchgängige Geometrie entstanden, in der jeder Raum erreichbar ist. Über den Binärbaum lassen sich Start und Endpunkte setzen, die nicht in direkt verbundenen Räumen liegen. Die Aufteilung des Raumes lässt es auch zu das Teilgebiete in der Visualisierung angehoben oder gesengt werden, damit der Dungeon auch in Dreidimensionen zu begehen ist.

3.2 Zellulare Automaten

3.2.1 Komplexe Systeme

Zellulare Automaten gehören zum Gebiet von Komplexen Systemen, es wird auch von der Modellierung und Simulation von Komplexen Systemen gesprochen.
Komplexität wird allgemein als Äquivalent zur Kompliziertheit gesehen. Eine Ansicht die sich in den letzten Jahrzehnten verändert hat und zu einer klaren Trennung der beiden Begriffe geführt hat. Ein kompliziertes System erscheint verworren und chaotisch, wird es jedoch in kleine und übersichtliche Teile zerlegt kann es schnell verstanden werden und erscheint simpel. Im Gegensatz dazu bei der Komplexität, erzeugen die Einzelteile in Zusammenarbeit neue Funktionen und das Betrachten der Einzelteile verrät nichts über das Ganze. Eine bekannte Bezeichnung ist die, dass etwas mehr ist als die Summe seiner Teile, man redet dabei auch von den emergenten Merkmalen der komplexen Systeme. Dies ist eine wichtige Eigenschaft, die gebraucht wird um zum Beispiel komplexes biologisches Verhalten zu simulieren. Genauer wird dies in den Erläuterungen der Einzelgebiete ersichtlich.

3.2.2 Ursprung

Zellulare Automaten wurden zum allerersten Mal von Stanislaw Ulam um das Jahr 1940 in Los Alamos beschrieben. Das Ziel war es ein Computersystem zu entwicklen, dass sich wie ein Lebewesen selbst reproduzieren kann. Dies machte er mit seinem Partner John von Neumann, der das Konzept in den 50er Jahren zu einem universellen Berechnungsmodell erweiterte. Im 1969 veröffentlichten Buch “Rechnender Raum“ von Konrad Zuse, beschrieb er die Annahme, dass das ganze Universum ein Ergebnis eines gigantischen Zellular Automaten sei. Große Beliebt- und Bekanntheit erreichten Zellulare Automaten 1970, durch das von John Horton Conway erstellte „Game of Life“, welches immer noch das beliebteste Beispiel für sie ist. In den letzten Jahrzehnten gewannen sie wieder an Interesse, bei ihrem Einsatz von Simulationen von Naturprozessen. Sie sollten mit möglichst einfachen Regeln die Natur nachahmen und dabei Ergebnisse erzeugen die Rückschlüsse auf das simulierte Model erlauben.

3.2.3 Definition

Zelluläre Automaten werden für die Modellierung von räumlich diskreten dynamischen Systemen eingesetzt. Deren Zellenzustand zum Zeitpunkt **t+1** allein von dem Zustand der Zelle und deren Nachbarn zum Zeitpunkt **t** abhängig ist.
Es handelt sich dabei um ein Zeit diskretes dynamisches System, was bedeutet das Zustandsänderungen in gleich groß bleibenden Zeitabständen stattfinden.
Ein Zellularer Automat ist durch folgende Eigenschaften definiert:

- ein Raum R (Zellraum)
- eine endliche Nachbarschaft N
- eine Zustandsmenge Q
- eine lokale Übergangsfunktion

3.2.3.1 Der Raum

Bei der Geometrie der Zellanordnungen gibt es eine Vielzahl von Kombinationen, die sich bei einer reinen theoretischen Anwendung räumlich unendlich ausbreiten können. Für eine praktische Nutzung wird dieser Raum natürlich begrenzt, was zu einer Bildung von Randbedingungen führt. Die Gitterstruktur aus dem der Raum zusammengesetzt wird kann aus verschiedenen geometrischen Formen bestehen. Die, vor allem in Beispielen, meist genutzte Geometrie ist die des Quadrats, die Verwendung anderer Strukturen wie Dreiecke oder die in der Natur vorkommenden Hexagone sind auch möglich. Weitere Muster wie die eines Kristallgitters oder die Kombination verschiedener Geometrien steht nichts im Weg, solange eine Parkettierung stattfindet. In ihrer dimensionalen Anordnung sind sie nicht begrenzt, von einem 1- bis zum n-dimensionalen Raum ist alles möglich.
Für die Visualisierung des Prozesses beschränkt man sich auf 1- bis 2-Dimensionen, denn schon ab der 3 Dimension bekommt man Probleme es anschaulich darzustellen und bei noch höheren Raumordnungen wird es unmöglich.

3.2.3.2 Die Nachbarschaft

Nach der Festlegung der Struktur der Zellen muss nun noch eine Regel definiert werden die festlegt, welche umliegenden Zellen Nachbarn sind. Diese werden von jedem Entwickler selbst definiert und müssen nur die Bedingungen der eigenen Anwendung erfüllen. Wenn nun die Nachbarschaft für den Automaten bestimmt wurde, ist sie auf alle Zellen anzuwenden. Zwei Fälle mit einer Quadratnischenstruktur im 2-Dimensionalen Raum sind vordefiniert. Erstens das alle Zellen die eine Verbindung über die Kanten oder Ecken haben als Nachbar gelten, dieses ist als Moore Nachbarschaft bekannt (Abb. 9). Wenn nur die als Nachbarn gelten, die nur eine Verbindung über Kanten haben ist das die Neumann Nachbarschaft (Abb. 10). Blau ist die aktive Zelle und Rot sind die als Nachbarn deklarierten Zellen. Die pinken Zellen in der von Neumann Nachbarschaft sollen zeigen, dass auch Zellen die mehrere Felder entfernt liegen, als Nachbar definiert werden können.

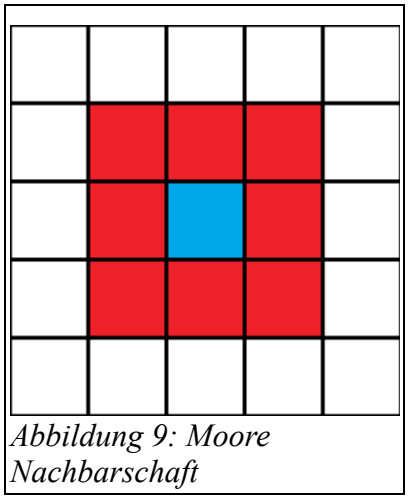

Abbildung 9: Moore Nachbarschaft

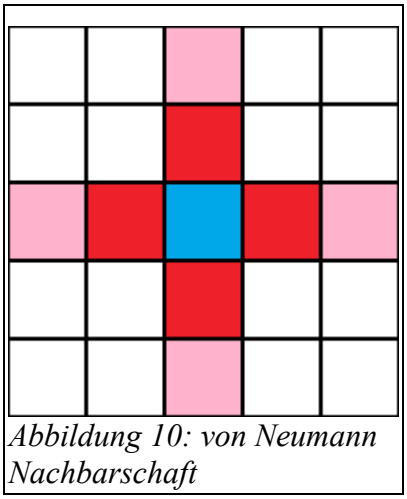

Abbildung 10: von Neumann Nachbarschaft

3.2.3.3 Die Zustandsmenge

Jede Zelle hat eine Menge an Zuständen, die sie einnehmen kann. Diese ist in den meisten fällen klein, da auch mit wenigen Zustandswerten hochkomplexe Systeme simuliert werden können. Wie viele es nun sind, hängt allein von der Problemstellung ab, die es zu lösen gilt. Die am häufigsten zu findende Zustandsmenge ist die binäre mit 1 oder 0, welche den Grundzustand einer Zelle mit lebendig oder tot darstellt.

3.2.3.4 Übergangsfunktion

Der große Variationsreichtum Zellularer Automaten ist auch den Regeln zuzurechnen, die das „Zusammenleben“ bzw. die Veränderungen der Zellen über die Zeit bestimmen. Sie legen fest, ob eine Zelle ihren Zustand in Abhängigkeit zu ihren Nachbarn ändert. Im klassischen Beispiel mit binären Zuständen bestimmt die Übergangsfunktion, ob eine Zelle am Leben bleibt, stirbt oder neu geboren wird. Gewählt werden die Übergangsfunktionen spezifisch für die Bedingungen, die man simulieren möchte. Die Anzahl aller möglichen Übergangsfunktionen lässt sich über die Anzahl der Zustände **k** und der Nachbarn **n** berechnen. Die Formel für die Berechnung lautet k^{k^n} . Das macht bei einer Mooreschen Nachbarschaft und Zwei zuständen schon 10^{77} Regeln.
Jede Regel entspricht dabei einem eigenen Automaten.

3.2.3.5 Randbedingungen

Für die praktische Anwendung muss der Zellraum begrenzt sein. Dies führt dazu, dass die Randzellen eine andere lokale Umgebung haben als die Zellen im Inneren. Bei einer Zelle die sich in einer Ecke befindet und eine Moore Nachbarschaft hat, sind nur noch drei der sonst acht Nachbarn vorhanden. Um mit dieser Problematik umzugehen, gibt es drei Strategien die offene, periodische und symmetrische Randbedingung.
In der offenen wird die Artefakt Bildung hingenommen, die durch die am Rand liegenden Zellen entstehen können.
Um diese unerwünschten Effekte zu vermeiden, können periodische Randbedingungen erstellt werden, beim Überschreiten des Randes eines Raumes kommt man auf der parallel liegenden Seite des Raumes an. Aus einem eindimensionalen Automaten würde man einen Ring formen. Damit wäre die linke Nachbarzellen der ersten Zelle, die letzte Zelle in der Reihe. Ein zweidimensionaler Automat wird mit diesem Verfahren zu einem Torus, ein Schlauchreifen bzw. ein Donut. In der 3 Dimension würde es ein Hypertorus werden, etwas was sich nur schwer darstellen lässt.
Der dritte Ansatz ist die der Symmetrie, die auch die Artefakt Entstehung verhindern soll. Hier wird durch Spiegelung der Raum künstlich über den Rand der Zelle vergrößert, der Rand selbst ist hier die Spiegelachse.
Sicherstellen können es die beiden Ansetze nicht, dass keine störenden Effekte auftreten. Das ist etwas was bei jedem neuen Automaten jeweils getestet werden muss.

3.2.4 Anwendung

Die allgemeine Beschreibung des Konzeptes findet hier mit dem „Spiel des Lebens“ statt.
Es soll gezeigt werden, wie die Eigenschaften aus der Definition angewendet werden.
Das spezifische Beispiel für ihre Nutzung in Videospielen ist das Spiel „Galak Z“ von 17-Bit aus dem Jahr (2015).

3.2.4.1 Das Spiel des Lebens

Im englischen als „Conway's Game of Life“ bekannt, ist eines der populärsten Beispiele für Zelluläre Automaten. Im Jahre 1970 erdachte John Horton Conway diesen 2-Dimensionalen Automaten mit einer Quadratischen Geometrie. Weiter bestimmte er die Regeln,
dass mit einer binären Zustandsmenge (0 = tot und 1 = lebendig) und einer Mooreschen Nachbarschaft zu arbeiten ist. Als Übergangsfunktion bestimmte er, dass bei weniger als 2 Nachbarn eine Zelle an Vereinsamung stirbt und mit mehr als 3 an Überbevölkerung eingeht. Sie bleibt am Leben, wenn sie genau 2 oder 3 Nachbarn besitzt und es werden neue geboren wenn eine tote Zelle 3 Nachbarn hat. Welche Zellen am Anfang leben, bestimmt der Spieler selbst. Dies trifft auf alle Zellularen Automaten zu, nicht nur auf das „Spiel des Lebens“.

Zustand	Nachbaranzahl	Folgezustand
0	2	0
1	2	1
0	< 2 oder > 3	0
1	< 2 oder > 3	0
0	3	1
1	3	1

Die Anwendung von diesem Zellularen Automaten sehen wir hier in Abb. 11. Die schwarz markierten Zellen sind die lebendigen und die weißen sind die toten Zellen. Die Zahlen sagen aus, wie viele Nachbarn jede einzelne Zelle hat und ihre Farbe sagt aus ob sie im folgenden Zyklus weiter lebt (grün) oder stirbt (rot). In den Feldern die durch eine hellblaue Färbung und einer grünen Zahl hervorgehoben werden, wird in der nächsten Iteration eine neue Zelle geboren. Auf diese Weise funktionieren alle Automaten, natürlich unter Berücksichtigung eigener Übergangsfunktionen.

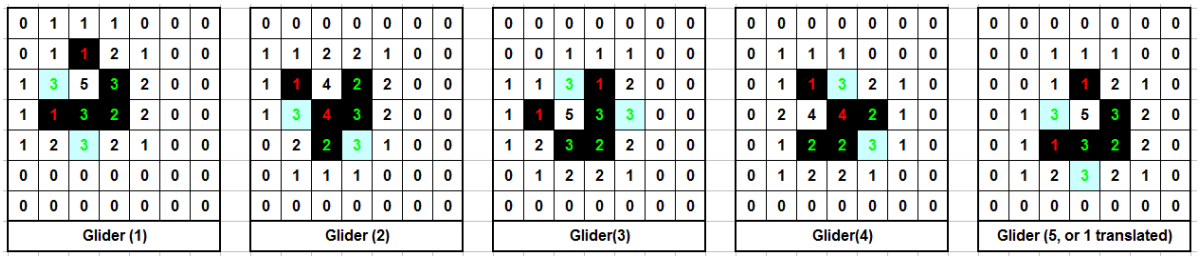

Abbildung 11: Glider (dt. Gleiter)

Dieses spezielle Beispiel, bekannt unter den Namen Gleiter, hat noch eine besondere Eigenschaft. Mit dem Anfangszustand von Glider(1) wird beim „Spiel des Lebens“ diese Zellanordnung in Glider(5) wieder repliziert, mit einer Versetzung um je eine Zelle nach unten und nach rechts. Das ist ein Vorgang, der bei einem endlosen Raum auch endlos weiterlaufen würde.

3.2.4.2 Galak-Z

Galak-Z: The Dimensionale ist ein Videospiel aus der Kategorie Roguelike, es besitzt also eine spielerische Anlehnung an den ursprünglichen Titel „Rogue“ von 1980. Entwickelt wurde es von dem Studio „17-BIT“ für die PlayStation 4 und ist im August diesen Jahres veröffentlicht worden. Zum Beginn der Entwicklung des Spieles stellten die Entwickler fest, dass der Zeitaufwand für die Erstellung der Dungeons per Hand zu groß wäre und sie entschlossen sich mit Prozeduralen Techniken zu helfen.
Aufgrund ihrer Rahmenbedingungen für die Dungeons, zweidimensionale verwickelte Höhlensysteme, erschienen Zellulare Automaten die sinnvollste zu nutzende Technik. Beim Erstellen dieser Dungeons fiel auf, wie in Abb. 12 zu sehen ist, dass sie eine interessante Raumstruktur generieren, die aber nicht dafür geeignet ist sie mit Inhalten zu füllen.
Das Platzieren von Gegnern oder Objekten würde mehr Zeit in Anspruch nehmen als es dauern würde es per Hand zu bauen.
Sie kamen zu dem Schluss, auch wenn ein Automat nicht den ganzen Dungeon erzeugen kann, können sie einzelne Räume produzieren, die zusammengesetzt ein Dungeon ergeben.
So ein Raum wie er in Abb. 13 zu sehen ist, wäre so ein Automat.

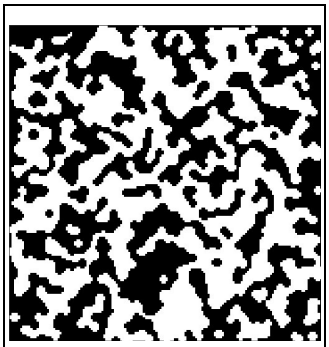

Abbildung 12: Zellulärer Automat für einen Dungeon

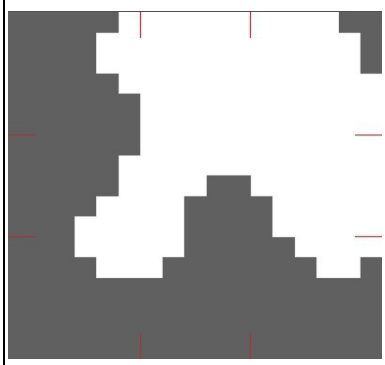

Abbildung 13: Zellulärer Automat für einen Raum

Dieser wurde um weitere Eigenschaften erweitert wie vordefinierte Ausgänge, verkleinerter Raum auf ein 17x17 Gitter und der Möglichkeit sie später individuell zu modifizieren.
Bevor die spezifischen Rahmenbedingungen festgelegt wurden, musste die Fragestellung der Generierung der Raumstruktur, in der die Automaten platziert werden, geklärt sein.
Sie probierten verschiedene Ansetze aus, darunter auch binäre Raumpartitionierung und kamen über Z-Kurven zu der Hilbert-Kurve. Beide sind sogenannte FASS-Kurven, was für „space-**f**illing, self-**a**voiding, **s**imple and self-**s**imilar“ steht. Diese sind in der Lage mit einer stetigen eindimensionalen Kurve einen zweidimensionalen Raum komplett auszufüllen, wie in Abb. 14 zu sehen ist. Damit nicht immer die selbe Raumstruktur vorhanden ist, wird eine Hilbert-Kurve dritten Grades genommen und ein 4x4 Gitter an einer beliebigen Position platziert. In Abb. 15 ist so ein Gitter zu sehen und der Raumverlauf des Dungeon ergibt über den Verlauf der Hilbert-Kurve innerhalb des Gitters. Dort sieht man auch den Fall, wenn die Kurve das Gitter verlässt und wieder eintritt. Die rot markierten Zellen werden entfernt oder von dem Dungeon Designern als Sackgassen für besondere Ziele verwendet. Der Prozess kann auch Strukturen erzeugen, die nicht benutzbar sind. Er ist aber durch seine geringe Laufzeit in der Lage im Sekundentakt neue Aufbauten von Dungeons zu generieren.

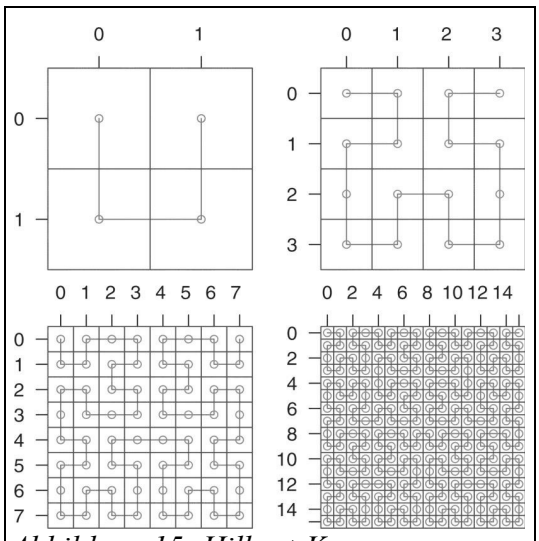

Abbildung 15: Hilbert-Kurve

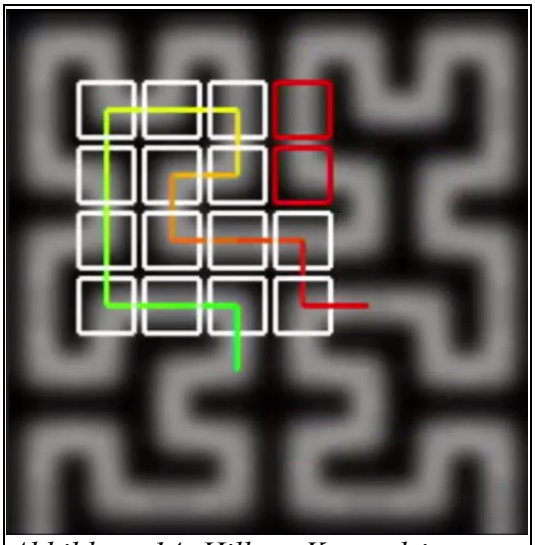
Abbildung 14: Hilbert-Kurve dritten Grades

So ein fertiger Dungeon ist in Abb. 16 zu sehen, man kann die Hilbert-Kurve wieder erkennen und die Zellulären Automaten haben interessante Räume erschaffen.
Die Befüllung der Welt mit Gegnern, Objekten und Effekten wurde auch zum großen Teil automatisiert, damit nur Feinkorrekturen ausgeführt werden mussten.

Abbildung 16: Galak-Z Dungeon Beispiel

3.3 Generative Grammatik

3.3.1 Ursprung

Begründet wurde die Generative Grammatik von Linguist Noam Chomsky im Jahre 1957 mit seinem Werk „Syntactic Structures“. Seine Idee lautete, dass es möglich sein sollte die Gesetzmäßigkeiten der menschlichen Sprache algorithmisch zu beschreiben.

3.3.2 Definition

Die von Chomsky formulierte Generative Grammatik hat folgende Komponente, die Grammatik $\boldsymbol{G} = (\boldsymbol{N}, \boldsymbol{T}, \boldsymbol{S}, \boldsymbol{P})$, Nichtterminalsymbole $\boldsymbol{N} = \{\boldsymbol{S}\}$, Terminale Symbole $\boldsymbol{T} = \{a, b, c\}$, Startsymbol $\boldsymbol{S}$ und die Produktionsregeln $\boldsymbol{P}$. Nichtterminalsymbole sind Symbole die nicht im entstehenden Wort vorkommen und durch die Produktionsregeln mit Terminal Symbolen ersetzt werden.

3.3.2.1 Chomsky's Modell

Chomsky's Konzept der Generativen Grammatik angewendet auf die menschliche Sprache, im Beispiel Englisch, würde vereinfacht so aussehen.
Wir haben $\boldsymbol{N} = \{\boldsymbol{S}, NP, VP, D, V, N\}$ mit einem Startsymbol $\boldsymbol{S}$ und die Satzbestandteile Nominalphrasen NP und Verbalphrasen VP**,** Determinieren D, Nomen N und Verben V. Diese Produktionsregeln, zu sehen in Abb. 17, sind der Versuch unsere Grammatikregeln algorithmisch darzustellen. Die Terminal Symbole sind $\boldsymbol{T} = \{the, dog, bone, ate\}$, welche den Satz bilden werden.

$$S \rightarrow NP\,,\ S \rightarrow VP$$
$$NP \rightarrow D\,,\ NP \rightarrow N$$
$$VP \rightarrow V\,,\ VP \rightarrow NP$$
$$D \rightarrow the\,,\ V \rightarrow ate$$
$$N \rightarrow dog\,,\ N \rightarrow bone$$

Abbildung 17: Produktionsregeln

Beim Ausführen der Regeln entsteht der in Abb. 18 zu sehende Syntax Baum, der sich verästelt bis keine Nichtterminalsymbole mehr vorhanden sind. Bei jedem Symbol bei dem es mehr als eine Ersetzungsregel gibt findet die Aufspaltung statt, weshalb der entstandene Satz „the dog ate the bone“ nicht das einzige valide Ergebnis ist. Sätze die nach dieser Grammatik zwar korrekt wären aber nach unser Logik nicht, sind „the bone ate the bone“ oder „the bone ate the dog“.

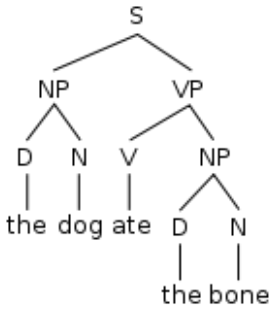

Abbildung 18: Syntax Baum

3.3.3 Anwendung

Wir haben gesehen wie Generative Grammatik für die Erstellung einer menschlichen Sprache eingesetzt wurde. Wenn man sich nun von diesem Ansatz löst und das Grundkonzept zusammen mit ihrem Nebenprodukt wie dem Syntax Baum betrachtet, fallen eine Vielzahl neuer Anwendungen auf. Da die praktische Arbeit mit Generativer Grammatik arbeitet, werden sie hier nur oberflächlich beschrieben.

3.3.3.1 Form Grammatik

Der Name verrät schon, dass es hier nicht mehr um das Bilden von Sätzen geht. Statt mit Symbolen, wie zum Beispiel aus dem Lateinischen Alphabet, sind hier die Elemente Punkte, Linien, Flächen und Volumen. Diese werden wie aus dem Grundkonzept bekannt mit ihren Produktionsregeln zu neuen Objekten geformt. In der Form Grammatik können die Produktionsregeln mehr als nur Ersetzen. Sie sind in der Lage Elemente zu addieren, subtrahieren, transformieren, rotieren oder zu reflektieren.
Die Abb. 19 zeigt eine vereinfachte Anwendung. Es lassen sich damit aber Komplexere Strukturen erzeugen, wie zum Beispiel Häuser.

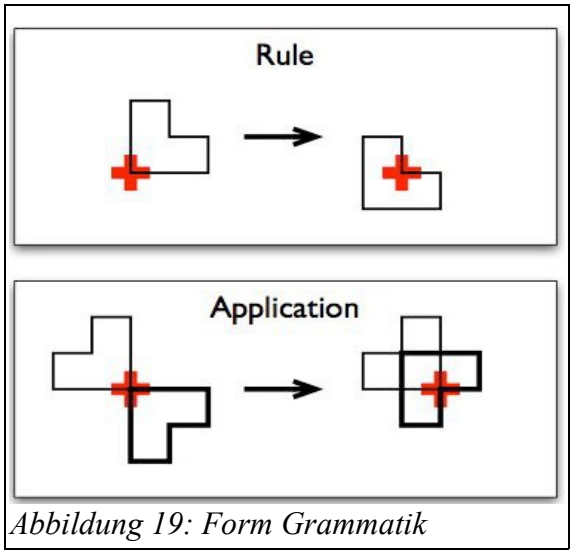

Abbildung 19: Form Grammatik

4 Die praktische Anwendungen

4.1 Technik

Zur Generierung eines Dungeons wird das Konzept der Generativen Grammatik verwendet. Es gab mehrere Gründe, welche zur Auswahl dieser Technik geführt haben, bestehend aus einem technischen Vorteil und persönlichem Interesse. Der technische Vorteil ist, dass es mit ihr möglich ist eine Dungeonstruktur durch eine Folge von Symbolen darzustellen. Mein persönlicher Grund ist die Faszination an Konzepten aus anderen Gebieten, die auf etwas Neues angewendet werden können. Das Darstellen einer Struktur durch eine Zeichenkette, die in diesem Kontext „seed“ genannt wird, in Deutsch „Samen“, ermöglicht das einfache speichern oder teilen eines Dungeons. So wird auch hier verfahren: Zuerst wird eine Buchstabenfolge die den Dungeon repräsentiert generiert und dann damit im Folgeschritt visualisiert wird. Der Unterschied zum Konzept von Chomsky ist, dass der entstehende „seed“ nicht aus allen Blättern des Syntaxbaumes zusammengesetzt wird. Stattdessen werden den Produktionsregeln, in den Fällen wo mehr als ein Terminales Symbol für ein Nichtterminalsymbol eingesetzt werden kann, Wahrscheinlichkeitswerte angeheftet, die entscheiden welche Regel angewendet wird. Das führt dazu, dass nur ein Blatt vorhanden ist, welches die Zeichenkette enthält. Anzumerken ist, dass in diesem Prozess nicht mehr determiniert ist was die Reproduzierbarkeit von „seed“ verhinderte, in diesem Projekt stellt das kein Problem dar.

4.2 Technologie

Für die Umsetzung und die Visualisierung der Prozeduralen Generierung wird das framework Unity3D eingesetzt, welches eine etablierte Videospiel-Engine ist. Sie wird neben Spielen auch für ein großes Spektrum von anderen Anwendungen aus dem Bereichen der 3D Visualisierung eingesetzt. Die verwendete Version der Software trägt die Nummer „5.0.1f1“ und ist vom 1. April 2015. Die Änderungen aus den neueren Iterationen haben keine Auswirkung auf dieses Projekt. Die vom Programm unterstützten Programmiersprachen sind JavaScript und C# und sind beide in der Umgebung gleich mächtig, aus persönlichen Präferenzen wird C# benutzt. Für die Visualisierung wir die Möglichkeit der Software genutzt sogenannte „prefabs“ zu erstellen, diese sind selbsterstellte Objektgruppen die als individuelles Objekt mehrfach in der Welt geladen werden können. In diesem Fall sind das vorgefertigte Objekte für die Dungeonräume und Korridore.

4.3 Ziel und Rahmenbedingung der Anwendung

Das Programm soll auf rudimentärer Weise die Struktur eines Dungeons generieren und sie Visualisieren. Damit dies als erfüllt gilt, wurden am Anfang der Arbeit einige Punkte genannt, die ein dreidimensionaler Dungeon als Grundvoraussetzung erfüllen muss. Ein Start und Endpunkt muss erkennbar sein und ein Mindestabstand dazwischen ist eine Voraussetzung. Die erzeugte Geometrie darf keine Lücken oder Löcher besitzen, weder in ihrer Struktur noch in der Darstellung durch Fehlplatzierung von Objekten. Es soll ein durchgehender Dungeon entstehen ohne isolierte Bereiche, keine Bildung von unerreichbaren Inseln. Der Verlauf der Räume soll über mehrere Ebenen gehen.

4.4 Die Theoretische Umsetzung

Nach der Generativen Grammatik ist folgendes zu definieren $\boldsymbol{G} = (\boldsymbol{N}, \boldsymbol{T}, \boldsymbol{S}, \boldsymbol{P})$ und deren Bedeutung zu erklären. Die Nichtterminalsymbole bestehen aus **N** = {**S**, A, Z, D, E, K}, **S** ist wieder das Startsymbol und die anderen sind Platzhalter für Dungeonelemente. Das „A“ ist für den Startpunkt und das „Z“ repräsentiert den Endpunkt. Mit „D“ wird ein Raum dargestellt und „E“ und „K“ stehen für Ebene und Korridor. Die Ebene gibt die Richtung an in der sich der nächste Raum befindet und der Korridor sagt aus ob, der Weg zwischen zwei Räumen nach oben, unten oder auf gleicher Höhe bleibt. Die Terminalen Symbole sind **T** = {a, b, c, d, i, j, k, l, m, n, z}, welche die spezifischen Eigenschaften eines Dungeons darstellen. In den Produktionsregeln **P,** aus Abb. 20, sieht man welche Nichtterminalsymbole sie ersetzen und die zu Beginn angesprochenen Wahrscheinlichkeitswerte. Sie bestimmen bei „D“ ob „b, c oder d“ eingesetzt wird, die jeweils verschiedene vorgefertigte Räume repräsentieren. Für die „E“ stellen „i, j und k“ die Entscheidung für den Weg nach links, die Mitte oder rechts da, mit einer sechzig prozentigen Wahrscheinlichkeit das „j“ bzw. Mitte gewählt wird. Dasselbe trifft auf „K“ zu, bei ihm wird mit derselben Chance „m“ ausgewählt, was einem gleichbleibenden Stockwerk entspricht. Die Symbole „l und n“ stehen für einen Korridor nach oben oder unten. Die anderen Symbole sind „a und z“, die für den Start- und Endraum stehen. Damit der Dungeon Generator variable große Strukturen erzeugen kann, ist die Produktionsregeln für **S** so angepasst, dass eine variable Anzahl von Räumen platziert werden kann. In Abb. 19 sieht man den Ausschnitt eines Syntaxbaumes, der bei dieser Grammatik entstehen würde.

$$S \rightarrow AD...DZ\,,\; A \rightarrow aEK$$
$$Z \rightarrow z\,,\; D \rightarrow bEK(0.33)$$
$$D \rightarrow cEK(0.33),\; D \rightarrow dEK(0.33)$$
$$E \rightarrow i(0.2),\; E \rightarrow j(0.6),$$
$$E \rightarrow k(0.2),\; K \rightarrow l(0.2),$$
$$K \rightarrow m(0.6),\; K \rightarrow n(0.2)$$

Abbildung 21: Produktionsregeln

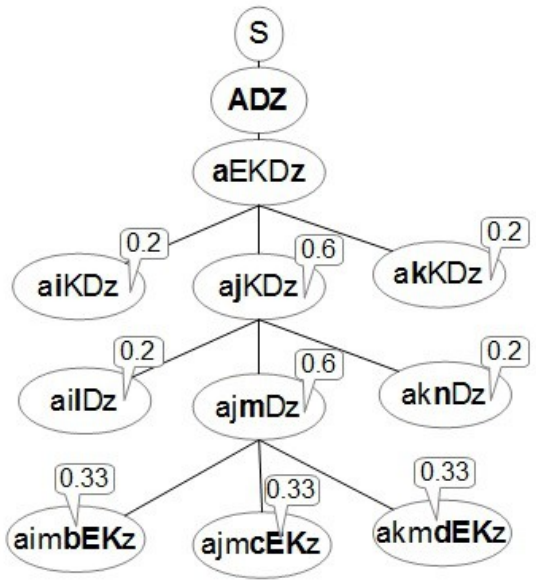

Abbildung 20: Syntax Baum Ausschnitt

4.5 Die Praktische Umsetzung

Hier sind zwei Vorgänge zu beschreiben, einmal der Prozess der „seed“ Generierung und seine Interpretierung für die Visualisierung des Dungeons.
Für die „seed“ Erstellung ist die Klasse „Grammatik“ verantwortlich. Die Zeichenkette, dargestellt durch einen „string“, wird Charakter für Charaktere durch geschritten und bei jedem wird kontrolliert ob und welche Produktionsregel anzuwenden ist. Für den Fall das mehr als eine Produktionsregel zutrifft, wird jedes mal eine Zufallswert zwischen 0 und 1 erzeugt. Diese Zahl bestimmt dann welche Regel angewendet wird. Wenn das Ende des „string“ erreicht ist, wird getestet ob sich noch Nichtterminalsymbol bzw. Großbuchstabe in ihm befindet. Trifft dies zu, wird der obere Schritt wiederholt bis dies der Fall ist.
Das Ergebnis ist ein „seed“ bestehend aus einer Zeichenkette mit Kleinbuchstaben.
Damit ist der erste Prozess abgeschlossen und der Schritt der Visualisierung kann beginnen. Mit diesem „seed“ wird nun die Klasse „generiertDungeon“ aufgerufen, die sich um die Visualisierung kümmert. Von der Funktion ähnelt sie der Erstellung der Zeichenkette, sie wird Charakter für Charaktere durchgegangen und überprüft, welches Objekt für dieses Symbol eingesetzt wird. Die Dungeonelemente werden in Unity3D mit einfachen Geometrischen Objekten dargestellt. Der Start- und Endpunkt werden durch eine weiße Platte repräsentiert. Die Dungeonräume, die im „seed“ als „b, c und d“ zu finden sind, werden mit einer roten, blauen und grünen Platte dargestellt. Für die Korridore gibt es zwei vorgefertigte Objekte, einmal das für den Weg auf gleicher Eben und ein zweites für den Weg nach oben oder unten.

Der zweite ist länger, da er nicht nur den Abstand auf der Ebene überwinden muss, sondern auch den Höhenunterschied.
Beim Vorgang der Visualisierung ist vor allem zu beachten, dass die Platzierung der Elemente korrekt ist. Um dies zu erreichen ist die richtige Interpretierung des „seed“ notwendig und das Vermeiden von Spezialfällen. Ein möglicher Dungeon der hier entstehen kann ist in Abb. 22 zusehen.

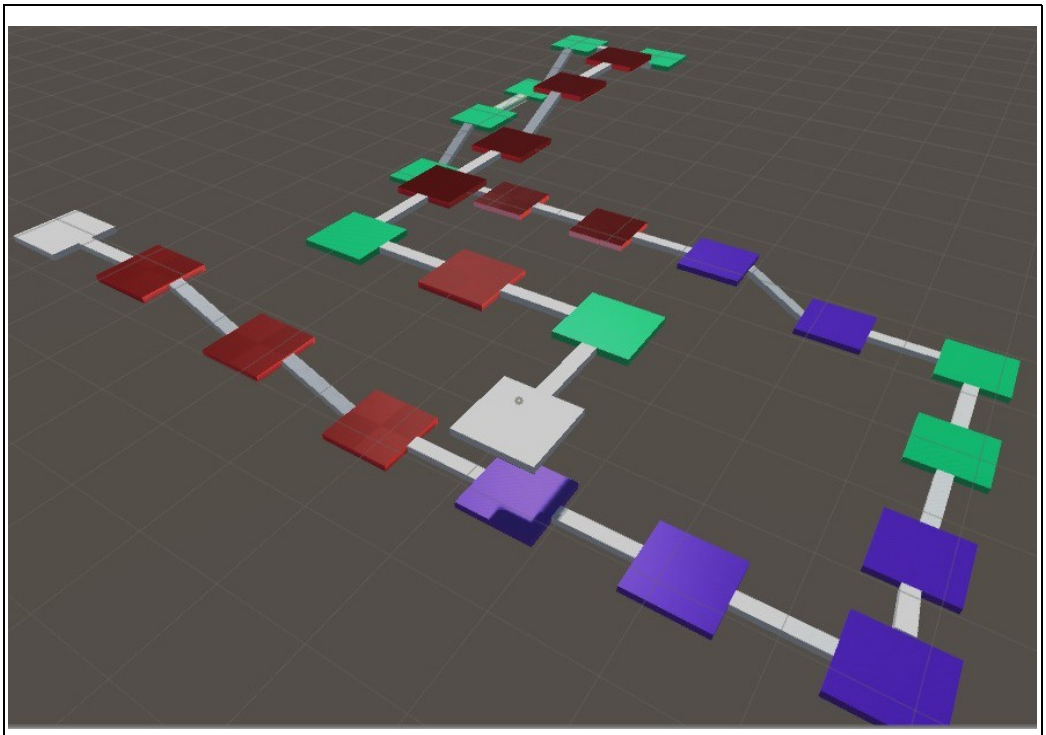

Abbildung 22: Ein möglicher Dungeon

Der Quellcode zu dieser Anwendung ist ausführlich kommentiert um das Verständnis für das Programm zu erhöhen.

5 Fazit

5.1 Techniken

Hier sollen nochmal die Vor- und Nachteile der beschriebenen Techniken im Einzelnen aufgeführt werden. Dabei soll weniger der Algorithmus selbst sondern das Produkt, welches er generiert, eine Bewertung erhalten.

5.1.1 Binäre Raumpartitionierung

Die am weitesten verbreitete Technik, die für das Generieren von Dungeons angewendet wird ist die binäre Raumpartitionierung. Zu Beginn lassen sich schon Rahmenbedingung für das Ergebnis festlegen, wie die Raumanzahl, der Maximale oder die Minimale Größe der Räume und die Gesamtfläche des Dungeons. Über den Binärbaum lassen sich einzelne oder in Gruppen zusammengefasste Bereiche eines Dungeons in der Ebene verändern oder auf andere Weise modifizieren. Nachteile gibt es bei dieser Technik nur wenn man die Typische Struktur des verfahren ändern möchte.

5.1.2 Zellulare Automaten

Zellulare Automaten bieten sich an, wenn man eine organisch gewachsene Struktur als Dungeon bekommen möchte, wie zum Beispiel bei einem Höhlensystem. Sie lassen sich schnell erzeugen und man bekommt direkt eine Visualisierung des Raumes.
Im Beispiel Galak-Z wurde aber gezeigt, dass ein einzelner Automat nicht sinnvoll eingesetzt werden kann. Dass sich die zusätzliche Arbeit für das Erzeugen einer Dungeonstruktur lohnen kann, wurde aber auch bewiesen.

5.1.3 Generative Grammatik

Die Generative Grammatik sticht im Vergleich zu den anderen Verfahren damit heraus, dass es sich um keinen Räumlichen Algorithmus handelt. Während bei den anderen eine Geometrische Struktur entsteht, die schon als Visualisierung genutzt werden kann, haben wir hier eine Symbolische Struktur. Das kann nachteilig sein, ermöglicht es aber auch Dungeonelemente mit spezifischen Eigenschaften schon in der Struktur zu generieren.

5.2 Anwendung

Das modifizierte Konzept der Generativen Grammatik produziert schöne Ergebnisse in der Anwendung. Die Anforderungen die für das Programm aus „Ziel und Rahmenbedingung“, wie Start- und Endpunkt oder vollständige Geometrie, werden erfühlt.
Die Eigenschaft, dass dieses Verfahren nicht determiniert ist, verursacht hier kein Problem, da immer neue geniert werden sollen und wenn es doch notwendig ist den selben Dungeon noch einmal zu erzeugen, kann der „seed“ von ihm gespeichert werden. Der Prozess der Visualisierung erzeugt beim selben „seed“ immer noch dasselbe Produkt.
Ein Nachteil ist die in der Generativen Grammatik fehlende Räumlichkeit, wie in 5.1.3 angesprochen. Anhand des „seed“ kann nicht mit bloßem Auge erkannt werden ob sich bei der Visualisierung die Geometrie kreuzen wird. Dies ist etwas was sich zwar prüfen lassen sollte, hier aber leider nicht mehr umgesetzt werden konnte.
Weiter Funktionen, die ein kompletter Dungeon Generator bräuchte, sind die Möglichkeiten die Maximale und Minimale Ausbreitung der Struktur vorher festzulegen, so das Zulassen von Verzweigungen um noch interessantere Ergebnisse zu erhalten.

6 Quellen

6.1 Textquellen

Prozedural Generierung

Mark Hendiikx, Sebastiaan Meijer, Joeri van der Velden, Alexandru Iosup: Procedural Content Generation for Games: A Survey, http://www.dingli.org/Website/GameAI_2/Readings/4_PCG.pdf letzter Zugriff 09.08.2015

Roland van der Linden, Ricardo Lopes, Rafael Bidarra: Procedural generation of dungeons, http://graphics.tudelft.nl/~rafa/myPapers/bidarra.TCIAIG.2014.pdf Zugriff 09.08.2015

Roland van der Linden, Ricardo Lopes, Rafael Bidarra: Designing Procedurally Generated Levels, https://graphics.tudelft.nl/Publications-new/2013/LLB13a/designingprocedurally.pdf letzter Zugriff 09.08.2015

Gillian Smith: The Future of Procedural Content Generation in Games, http://www.exag.org/papers/exag2014_submission_7.pdf letzter Zugriff 09.08.2015

Binäre Raumpartitionierung

Matthias Neumann, Anna Seufert: Binäre Raumpartitionierung: https://wuecampus2.uni-wuerzburg.de/moodle/pluginfile.php/194624/mod_resource/content/1/BRP_Pr%C3%A4sentation.pdf letzter Zugriff 14.08.2015

Cristoph Göth, Binary Space Partitioning Trees: http://www.uni-koblenz.de/~cg/veranst/ws0001/sem/Goeth.pdf letzter Zugriff 14.08.2015

Basic BSP Dungeon generation: http://www.roguebasin.com/index.php?title=Basic_BSP_Dungeon_generation letzter Zugriff 14.08.2015

Zellularen Automaten

Svacko, Vladimir: Seminar Arbeit Zelluläre Automaten: https://www.matse.itc.rwth-aachen.de/dienste/public/show_document.php?id=7297 letzter Zugriff 09.08.2015

Lawrence Johnson, Georgios N. Yannakakis, Julian Togelius: Cellular automata for real-time generation of infinite cave levels: http://julian.togelius.com/Johnson2010Cellular.pdf letzter Zugriff 09.08.2015

Buckmann, Hans-Georg: Zelluläre Automaten: http://www.vlin.de/material/ZAutomaten.pdf letzter Zugriff 09.08.2015

Aikman, Zach: Galak-Z, Forever: Building Space-Dungeons Organiclally: Präsentation http://twvideo01.ubm-us.net/o1/vault/gdc2015/presentations/Aikman_Zach_Galak-Z,%20Forever.pdf Vortag: https://www.youtube.com/watch?v=ySTpjT6JYFU letzter Zugriff 09.08.2015

Generative Grammatik

Einführung in die Grundlagen der Generativen Grammatik: http://tu-dresden.de/die_tu_dresden/fakultaeten/philosophische_fakultaet/iph/thph/braeuer/lehre/putnam_bedeutung/Generative%20Grammatik.pdf letzter Zugriff 12.08.2015

Simon Colton, Alison Pease: Visual Grammars and some Applications: http://ccg.doc.gold.ac.uk/teaching/ludic_computing/ludic5_6.pdf letzter Zugriff 12.08.2015

Grace, Kazjon: Shape Grammas http://web.arch.usyd.edu.au/~kgra7942/DECO2013/L3.ShapeGrammars.pdf letzter Zugriff 14.08.2015

Anwendung

Showcase der Unity3D-Engine: https://unity3d.com/showcase, letzter Zugriff 12.08.2015

Archive der verschiedenen Unity3D Versionen: https://unity3d.com/get-unity/download/archive, letzter Zugriff 12.08.2015

6.2 Bildquellen

Prozedural Generierung

Rescue on Fractalus Wikipedia https://en.wikipedia.org/wiki/Rescue_on_Fractalus#/media/File:A5200_Rescue_On_Fractalus.png, Abb. 1, Public domain, letzter Zugriff 09.08.2015

Rogue Wikipedia https://en.wikipedia.org/wiki/File:Rogue_Unix_Screenshot_CAR.PNG , Abb. 1, Public domain, letzter Zugriff 09.08.2015

Elite https://upload.wikimedia.org/wikipedia/en/c/c4/BBC_Micro_Elite_screenshot.png, Abb. 1, fair use, letzter Zugriff 09.08.2015

Binäre Raumpartitionierung

Binäre Raumpartitionierung, Abb. 2, selbst erstellt

Fertig aufgeteilter Raum http://www.roguebasin.com/index.php?title=Basic_BSP_Dungeon_generation fair use, Abb. 3, letzter Zugriff 12.08.2015

Binärbaum zu Abb. 3, Abb. 4, selbst erstellt

Partitionierter Raum gefühlt mit Rechtecken http://www.roguebasin.com/index.php?title=Basic_BSP_Dungeon_generation fair use, Abb. 5, letzter Zugriff 12.08.2015

Verbindung der untersten Blätter über ihre inneren Knoten http://www.roguebasin.com/index.php?title=Basic_BSP_Dungeon_generation fair use, Abb. 6, letzter Zugriff 12.08.2015

zweite Iteration der Korridor Platzierung http://www.roguebasin.com/index.php?title=Basic_BSP_Dungeon_generation fair use, Abb. 7, letzter Zugriff 12.08.2015

Alle Teilräumen zu einem Verbunden http://www.roguebasin.com/index.php?title=Basic_BSP_Dungeon_generation fair use, Abb. 8, letzter Zugriff 12.08.2015

Zellularen Automaten

Moore-Nachbarschaft https://upload.wikimedia.org/wikipedia/en/d/d2/CA-Moore.png, Abb. 9, Creative Commons (CC-Zero) letzter Zugriff 09.08.2015

von Neumann-Nachbarschaft
https://upload.wikimedia.org/wikipedia/en/5/56/CA-von-Neumann.png, Abb. 10, Creative Commons (CC-Zero), letzter Zugriff 09.08.2015

Glider http://www.paleotechnologist.net/wp-content/uploads/2010/10/glider.png, Abb. 11, Public domain, letzter Zugriff 09.08.2015

Zelluläre Automaten für einen Dungeon: http://twvideo01.ubm-us.net/o1/vault/gdc2015/presentations/Aikman_Zach_Galak-Z,%20Forever.pdf, Abb. 12, Creative Commons,letzter Zugriff 10.08.2015

Zelluläre Automaten für einen Raum: http://twvideo01.ubm-us.net/o1/vault/gdc2015/presentations/Aikman_Zach_Galak-Z,%20Forever.pdf, Abb. 13, Creative Commons, letzter Zugriff 10.08.2015

Hilber-Kurve: http://twvideo01.ubm-us.net/o1/vault/gdc2015/presentations/Aikman_Zach_Galak-Z,%20Forever.pdf, Abb. 14, Creative Commons, letzter Zugriff 10.08.2015

Hilber-Kurve dritten Grades: http://twvideo01.ubm-us.net/o1/vault/gdc2015/presentations/Aikman_Zach_Galak-Z,%20Forever.pdf, Abb. 15, Creative Commons, letzter Zugriff 11.08.2015

Galak-Z Dungeon Beispiel: http://twvideo01.ubm-us.net/o1/vault/gdc2015/presentations/Aikman_Zach_Galak-Z,%20Forever.pdf, Abb. 16, Creative Commons, letzter Zugriff 11.08.2015

Generative Grammatik

Produktionsregeln Abb. 17, selbst erstellt

Syntax Baum
https://en.wikipedia.org/wiki/Generative_grammar#/media/File:Basic_english_syntax_tree.svg , Abb. 18 , Public domain, letzter Zugriff 09.08.2015

Form Grammatik
http://web.arch.usyd.edu.au/~kgra7942/DECO2013/L3.ShapeGrammars.pdf, Abb. 19 , Creative Commons, letzter Zugriff 14.08.2015

Anwendung

ProduktionsregelnAbb. 20, selbst erstellt
Syntax Baum Ausschnitt Abb. 21, selbst erstellt
Ein möglicher Dungeon Abb. 22, selbst erstellt