Designing Game Worlds

Kohärenz in der Spielweltgestaltung von Open-World-Games durch prozedurale Generierungstechniken

Mihajlo Nenad

Institut
Integrative
Gestaltung
Masterstudio

Vorgelegt von: Mihajlo Nenad
mail@mnenad.com

Betreut von: Ralf Michel, Réne Bauer

2018

Aus Gründen der besseren Lesbarkeit wird auf die gleichzeitige Verwendung männlicher und weiblicher Sprachformen verzichtet. Sämtliche Personenbezeichnungen gelten gleichermaßen für beiderlei Geschlecht.

Abstract

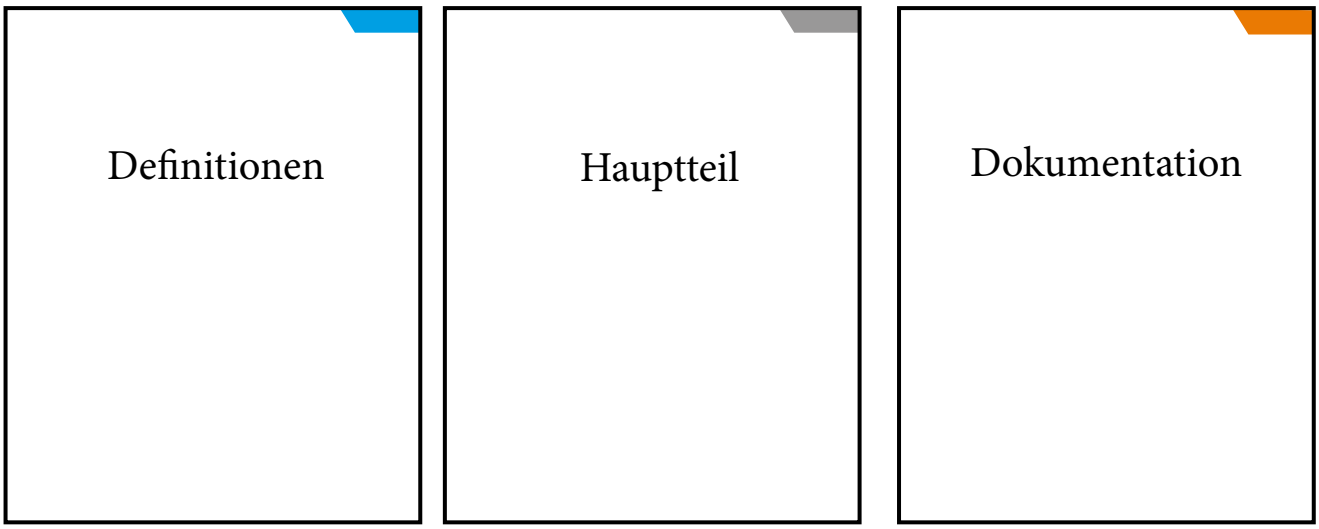
Sogenannte „Open-World-Games“ haben durch die schiere Grösse ihrer frei begehbaren Spielwelt das Potential für unzählige Stunden an Exploration und Spielspass. An solchen Produktionen arbeiten in der Regel grosse Teams ausgebildeter Spezialisten, die Spielinhalte designen, programmieren und sinnvoll in die Gamewelt integrieren. Eine Alternative zur herkömmlichen Gestaltung von Open-World-Games bietet die prozedurale Generierung der Inhalte durch Computersoftware. Verschiedene Spieltitel nutzen diese Generierung, um digitale Universen zu kreieren. Dabei können Probleme in der Kohärenz dieser Spielwelt auftreten, die die Spielerfahrung negativ beeinflussen können.

Diese Arbeit entstand aus der Motivation heraus, als Einzelperson ein Open-World-Game zu designen und dabei einen möglichst hohen Grad gestalterischer Kohärenz zu erreichen.

Inhaltsverzeichnis

1. Einführung

Diese schriftliche Arbeit bildet zusammen mit dem praktisch erarbeiteten Artefakt, in Form des Game-Prototyps *GenoTerra* (siehe Kapitel 6), meine Masterarbeit an der Hochschule für Gestaltung und Kunst Basel (FHNW) im Fachbereichs Design[1]. Das Ziel der Arbeit ist es, anhand des Prototypen Methoden zur Erzeugung von kohärenten Bildsprachen in Open-World-Games (und deren Komponenten) zu erarbeiten und konkret anzuwenden. Die Arbeit ist dabei in drei farblich gekennzeichnete Bereiche unterteilt:



Es ist essentiell für den Hauptteil (grau), sich der Definitionen der Begriffe (blau) bewusst zu sein, da viele der Begriffe aus dem Englischen entlehnt sind, bzw. viele technische Begriffe mehrdeutig interpretierbar sind. Die Dokumentation (orange) bietet zusätzliche Einblicke in den spezifischen Arbeitsprozess oder verweist auf externe Quellen zur vertieften Auseinandersetzung mit Inhalten dieser Arbeit.

[1] https://www.masterstudiodesign.ch

1.1 Definition - Game (Video-, Computerspiel)

Game: Zielorientiertes und regelbasiertes, digitales Spiel, welches auf visuell-interaktiver Basis auf einem Gerät (Computer, Spielkonsole, mobile Geräte, etc.) gespielt wird. Video-, und Computerspiele sind Games.

Videospiel: Game welches auf einem Gerät, auf Basis eines visuellen Outputs gespielt wird. Bsp.: Computer mit Bildschirm, Spielkonsole mit TV-Gerät, Smartphone, etc.

Computerspiel: Videospiele mit ausschliesslich Computern als Zielplattfrom.

Play: Nicht zielorientiertes, digital-interaktives Erlebnis, welches wie Games auf einem Gerät (Computer, Spielkonsole, mobile Geräte, etc.) genutzt wird. Eher eine spielerische Erfahrung als ein Spiel an sich.

Play + Regelwerk = Game

Der Terminus „Game“ entspringt dem Englischen und steht, ins Deutsche übersetzt, schlicht für „Spiel“. Umgangssprachlich wird „Game“ im Deutschen aber speziell für digitale Spielprogramme genutzt. Diese können einerseits auf einem Computer (Computerspiel/Videospiel), einer elektronischen Spielkonsole oder einem mobilen Gerät (Videospiele/ Mobile Games) ausgeführt werden. Videospiele sind in den meisten Fällen auf visueller Basis interaktiv. D.h. der Spieler reagiert über eine Schnittstelle (Controller, Touchscreen, etc.) auf visuelle Eindrücke, die von einem Bildschirm oder einem Display dargestellt werden.

Der Begriff des „Computers“ steht in dieser Arbeit für gewöhnliche Heimcomputer wie sie nach der aktuellsten Erhebung des Bundesamtes für Statistik aus dem Jahre 2014 bereits in fast 90% der Schweizer Haushalte zu finden waren.[2] Spielkonsolen bestehen, wie Computer auch, aus Elektronik-, und Kunststoffbauteilen (Hardware) sowie diversen digitalen Programmen zur Ausführung der Aufgaben des Gerätes (Software). 2014 besassen über 20% der Schweizer Haushalte mindestens eine Spielkonsole.[3] Spielkonsolen dienen dabei, in der Regel anders als Computer, fast nur dem Zweck des Ausführens von Spielprogrammen. Heute oft vorkommende Ausnahmen sind das Lesen und Abspielen von Musik oder Filmdateien durch Spielkonsolen, die Möglichkeit mit anderen Spielern über eine Internetverbindung zu kommunizieren (Live-Chat) und aufgenommene Video-, und Toninhalte live im Internet zu senden oder zu empfangen (Streaming).

Die erwähnte Unterscheidung zwischen den Begriffen „Computerspiel“ und „Videospiel“ wird im alltäglichen Sprachgebrauch ausserhalb der Spielerszene nicht immer beachtet. Spieler von digitalen Spielen unterscheiden jedoch sehr wohl zwischen den zwei Versionen oder verwenden „Game“ als Überbegriff. In dieser Arbeit wird der Begriffe „Game“ und „Spiel“ zur besseren Lesbarkeit im selben Sinne wie Video-, oder Computerspiel, also dem, eines digitalen Spielprogrammes, verwendet.

---

[2] Schweizer Bundesamt für Statistik: IKT-Ausstattung der Schweizer Haushalte nach Produktart, 2014. https://www.bfs.admin.ch/bfs/de/home/statistiken/kultur-medien-informationsgesellschaft-sport/informationsgesellschaft/gesamtindikatoren/haushalte-bevoelkerung/ikt-ausstattung.html (12.05.2018)

[3] Schweizer Bundesamt für Statistik: IKT-Ausstattung der Schweizer Haushalte nach Produktart, 2014. https://www.bfs.admin.ch/bfs/de/home/statistiken/kultur-medien-informationsgesellschaft-sport/informationsgesellschaft/gesamtindikatoren/haushalte-bevoelkerung/ikt-ausstattung.html (12.05.2018)

Auch die Begriffe „Gamerin“ oder „Gamer“ sind im Deutschen längst geläufig und beschreiben Personen, die gerne und häufig Games konsumieren. „Gamen“ als Verb wird als Pendant zu „Videospiele spielen“ verwendet. Besonders hier werden Unterschiede zum Englischen feststellbar. Im englischen Sprachgebrauch steht der Begriff „gaming“ einerseits für „Videospiele spielen“, kann aber genauso für nicht-digitale Spiele wie Kartenspiele oder Sport (physical gaming) genutzt werden. „Gaming“ wird in Grossbritannien auch als Umschreibung für „legales Glücksspiel“ genutzt.[4] Ein geläufigerer Ausdruck um das Nutzen von Videospielen in englischer Sprache auszudrücken wäre „playing video games“. Spieleentwickler bezeichnet man als „Gamedesigner“, wobei es zu beachten gilt, dass Gamedesigner bei der Produktion von Videospielen das theoretische Gesamtkonzept und dessen Regelwerk entwickeln und nicht etwa nur das visuelle Erscheinungsbild.

Allen Games ist gemein, das sie auf vorgegebenen Regeln basieren, die im Quellcode der Software eingebettet sind, und der spielenden Person das Spielziel klar vermitteln. Diese Regeln besagen, wie ein Game gewonnen, also das Spielziel erreicht, aber auch verloren, bzw. das Spielziel verfehlt werden kann.

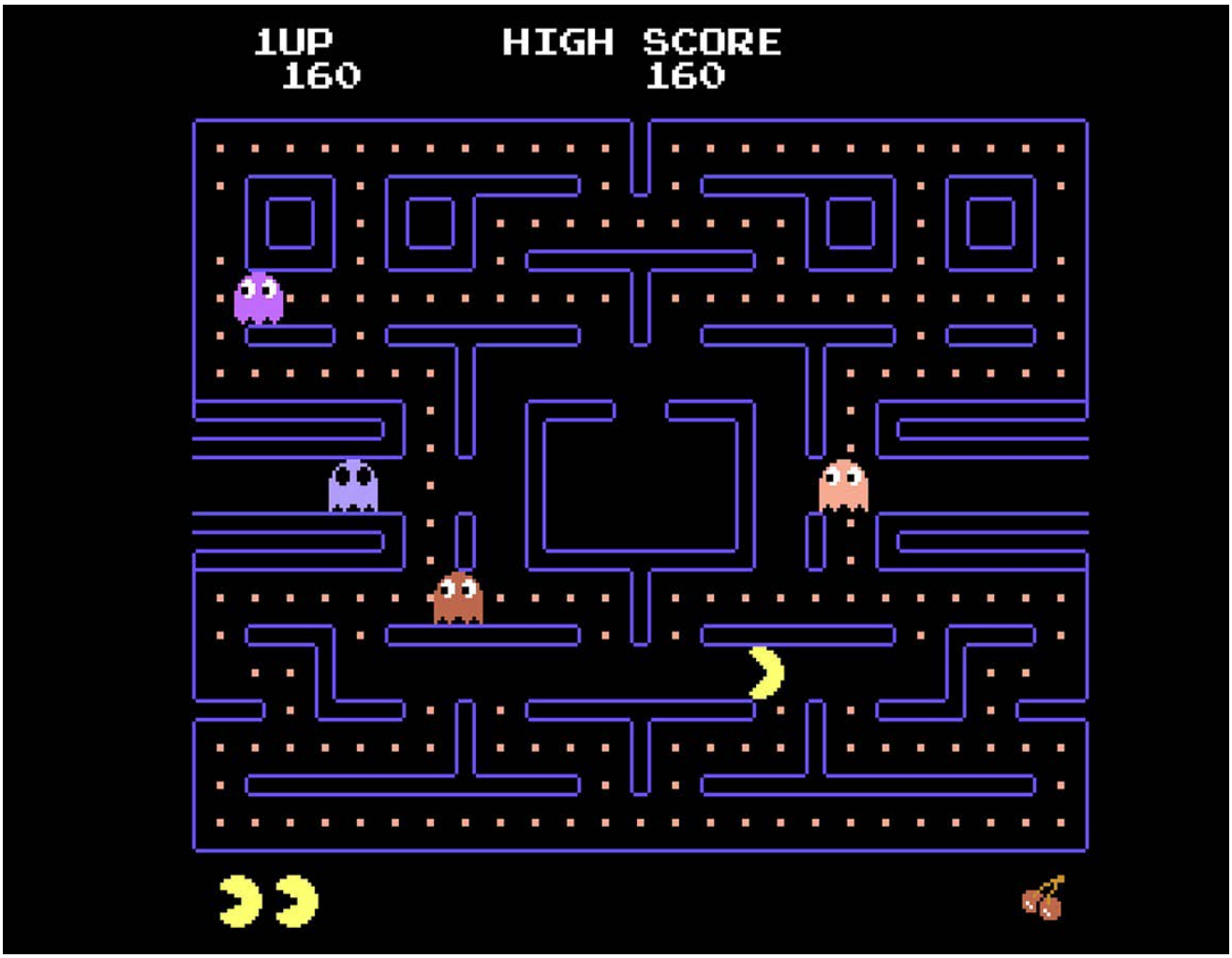


Bild 1: Eine von vielen Varianten des Spiels Pac-man

[4] Camebridge Dictionary: Meaning of „gaming“ in the English Dictionary. https://dictionary.cambridge.org/dictionary/english/gaming (12.05.2018)

Nimmt man das 2D-Spiel *Pac-man* [1] so kann man bei fast allen klassischen Versionen des Spiels folgende Regeln feststellen:

- **Ziel:** Genug Punkte sammeln um den aktuellen Highscore[5] zu überschreiten.

- **Siegkriterium (per Level):** Alle gelben Punkte müssen von Pac-man (durch Spieler gesteuert) gefressen werden. Falls dies erfüllt ist, startet als Belohnung ein höheres und herausfordernderes Level.

- **Niederlagekriterium:** Wird Pac-man von einem Geist berührt, verliert der Spieler eines von anfänglich drei Leben. Sind alle Leben aufgebraucht endet das Spiel und der Highscore wird der Rangliste hinzugefügt (falls hoch genug).

- **Spezialregeln:** In vielen Pac-man-Varianten kann der Spieler auch sporadisch auftretende Früchte in der Welt sammeln, die besonders viele Punkte geben. Zudem können auch die Geister unter bestimmten Bedingungen für kurze Zeit gejagt und gefressen werden.

1.1.1 Game vs. Play

Weist ein Spiel keine konkreten Regeln auf und baut stattdessen auf einer reinen Erfahrung des Nutzers im digitalen Raum, spricht man nicht mehr von einem „Game". Der korrekte Begriff lautet dann: „Play". Um diesen Unterschied zu verdeutlichen, hier einige Zitate von Persönlichkeiten, die sich mit den sinngemässen Begriffen „Game" und „Play" auseinandergesetzt haben.
Der 1945 verstorbene niederländische Kulturhistoriker Johan Huizinga definiert Spiel beispielsweise folgendermassen:

> "Spiel ist eine freiwillige Handlung oder Beschäftigung, die innerhalb gewisser festgesetzter Grenzen von Zeit und Raum nach freiwillig angenommenen, aber unbedingt bindenden Regeln verrichtet wird [...]" [6]

Der Kanadier und Schöpfer der berühmten Spielereihe *Civilization* [2], Sid Meier, zu Games:

„A game is a series of meaningful choices." [7]

Bild 2: Civilization 1 (1991)

[5] Highscore = Rangliste, die jeweils vom Spieler mit der höchsten Punktzahl angeführt wird.

[6] Huizinga, Johan: Homo Ludens. Reinbek bei Hamburg: Reclam (2001). S.3

[7] Rollings Andrew und Morris Dave: Game Architecture and Design. New Edition. Berkely Kalifornien: New Riders (2004) S.61

Eine Beschreibung des Spielprinzips von *Civilization* wurde Wikipedia entnommen, da die Autoren von Game-Wikis sich in der Regel selbst sehr stark mit den Spielen auseinandersetzen und verlässliche Quellen bilden. Sie lautet:

> „Ziel des Spieles ist es, andere Zivilisationen durch militärische Stärke oder kulturellen oder wissenschaftlichen Vorsprung zu übertreffen. Eine Siegmöglichkeit besteht somit in der Vernichtung sämtlicher gegnerischer Zivilisationen. Das Spiel kann aber auch durch das Bauen eines Raumschiffes, das erfolgreich nach Alpha Centauri fliegt, gewonnen werden. Dies setzt voraus, dass der Spieler eher auf den technischen Fortschritt setzt als auf militärische Stärke. Letztlich kann der Spieler auf eine hohe Gesamtwertung setzen, die das Spiel vergibt, wenn eine gewisse Spieldauer erreicht und das Spiel beendet wird. Hierbei können auch kulturelle Errungenschaften, wie der Bau von Weltwundern, eine Rolle spielen.“ [8]

Die vorangegangenen Zitate machen klar, dass ein solides Regelwerk und klar definierte Folgen von Spieleraktionen auf das Spielziel für eine Begriffsdefinition essentiell sind. Nur so kann die Tätigkeit im analogen Bereich als Spiel und im digitalen Bereich als Game bezeichnen werden. Im Gegensatz dazu steht das freie, nicht zielorientierte „Play“, welches im Deutschen missverständlich ebenso als „Spiel“ übersetzt werden kann.

Der spanische Philosoph, Schriftsteller und Literaturkritiker George Santayana beschreibt in seinem Werk „The Sense of Beauty“ den Begriff des „Play“ sinngemäss wie folgt:

> „Play is whatever is done spontaneously and for its own sake.” [9]

David O’Reilly überzeugte mit seinem Titel *Everything* [3], welcher im April 2017 erschien und auf verschiedenen Plattformen (Mac, PC, Linux, PS4)[10] spielbar ist, breite Massen der Gamer-, und Nicht-Gamer-Szene. David O‘Reilly‘s digitale Arbeiten werden in dieser Thesis unter 1.6.2 zur Erläuterung weiterer Sachverhalte im Bezug zu Kohärenz dienen. Ich beziehe *Everything* hier klärend ein, um es exemplarisch für die Abgrenzung zwischen Game und Play zu nutzen. Die Beschreibung von O’Reillys Werk auf der Videospielverkaufsplattform Steam lautet:

> „Be the Universe in this epic reality simulation game”
>
> ---
>
> „Everything ist ein interaktives Erlebnis, das dich alles sein lässt was du sein möchtest. Egal ob Tier, Planet, Galaxy. Reise zwischen dem Mikro-, und Makrokosmos hin und her und erforsche ein grosses, miteinander verbundenes Universum ohne verpflichtende Ziele oder Aufgaben.“ [11]

---

[8] Wikipedia. Begriff: Civilization (Computerspiel) https://de.wikipedia.org/wiki/Civilization_(Computerspiel) (30.05.2018)

[9] Santayana, George: The Sense of Beauty. Being the Outline of Aesthetic Theory. New York: Dover Publications, Inc. (1955) S.19

[10] Mac = Computer der Firma Apple mit dem sogenannten OSX Betriebssystem
PC = Personal Computer. Computer mit grösstenteils Windows Betriebssystemen der Firma Microsoft
Linux = Betriebssystem-Kernel (Alternatives Betriebssystem für Computer mit kostenlos zugänglichem Quellcode)
PS4 = Play Station 4. Spielkonsole der Firma Sony Computer Entertainment

[11] https://store.steampowered.com/app/582270/Everything/ (12.05.2018)

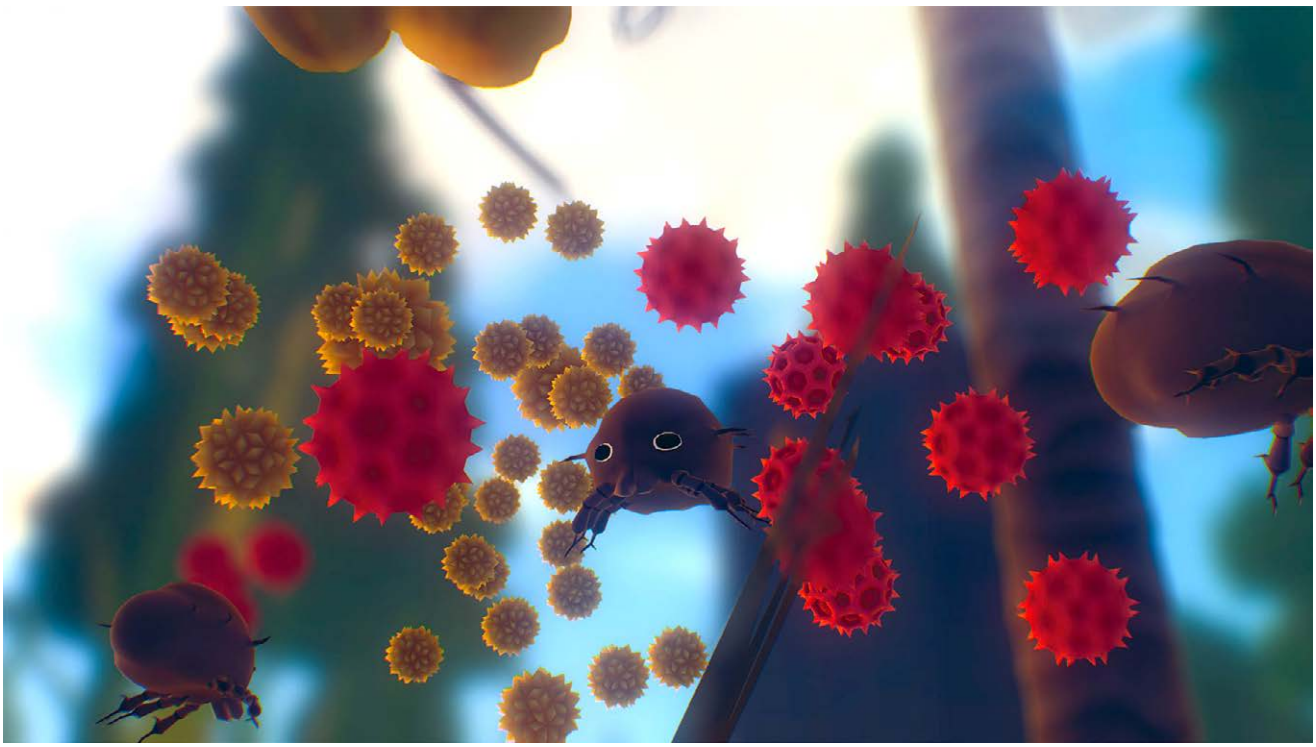


Bild 3: Ausschnitt aus Everything. Der User befindet sich gerade in der Rolle von Kleinstorganismen und sieht die Welt aus deren Perspektive

Meiner Meinung nach wird *Everything* fälschlicherweise oftmals als Game bezeichnet. Narrativer Inhalt der Software ist, wie zuvor beschrieben, in beliebige Bestandteile eines digitalen Universums zu schlüpfen und die Umgebung aus dessen Perspektive wahrzunehmen. Dabei werden an gewissen Punkten, im Verlauf der Reise des Users, Audioaufnahmen mit philosophischen Inhalten des englischen Religionsphilosophen Allan Watts eingespielt. Das ziellose, versunkene Umherwandeln in der digitalen Welt kann zusammen mit den Audioschnipseln eine durchaus emotionale Erfahrung erzeugen. Dies ist für ein Play, aber auch ein Game wünschenswert. Jedoch hat die Interaktion in *Everything* auf den Spielverlauf weder positiven, noch negativen Einfluss. Ein Spielziel fehlt, wie dem beschreibenden Zitat zu entnehmen war, komplett. Dies grenzt das digitale „Erlebnis“ eindeutig vom Begriff des zielorientierten Games ab.

1.1.2 Definition - Spielewelt

Um den Prototyp aus Kapitel 6 und dessen Argumentation im Aufbau der Spielewelt zu verstehen, muss der Begriff der „Welt“ geklärt sein.

Die Spielewelt eines Videospiels beinhaltet alle digitalen Inhalte, mit denen der Avatar[12] des Spielers im Kontext des Games interagieren, oder die der Spieler selbst wahrnehmen kann. Sei es nun visuell (Landschaften, Gegner, etc.) oder auditiv (Sound). Bei analogen Brettspielen kommen zudem die haptisch erfahrbaren Spielsteine hinzu, wobei Würfel oder bei digitalen Spielen der Computer selbst nicht zur Spielwelt gehören. Die Spielwelt bildet dabei ein zumeist fiktives Universum mit eigenen Regeln.

Zur Untermauerung und Vervollständigung dieser eigenen Definition, folgt noch eine weitere von Gunter Rehfeld, Autor des Handbuches „Game Design und Produktion“.

[12] Avatar = Künstliche Darstellung einer Person oder eines Wesens, das den Spieler in der virtuellen Welt repräsentiert.

„Der Spielraum definiert Auswahl und Anordnung von Spielelementen im Raum. Hier muss zwischen feststehenden, fixen Gamebits (einzelne Spielelemente) und den beweglichen, variablen unterschieden werden.“ [13]

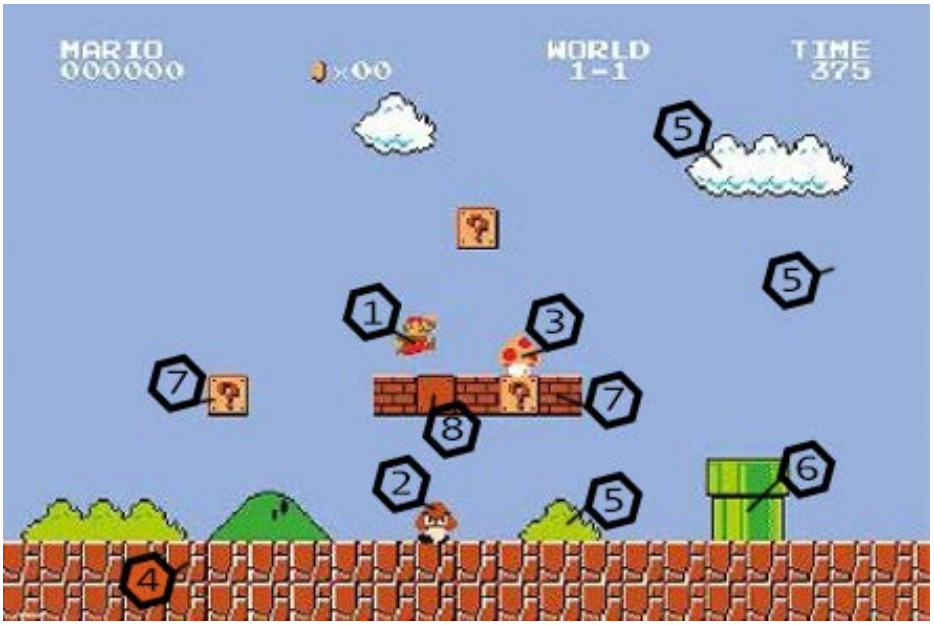


Bild 4: Super Mario Bros. (1985 in Japan / 1987 in Europa/USA veröffentlicht)

In *Super Mario Bros.*[4] bewegen sich der Spieler als Mario (1), sowie der Gegner Gumba (2) und der Pilz als Ressource (3). Sie sind variable Elemente, bzw. Gamebits[14] in der Spielewelt. Der Boden (4), der 2D-Hintergrund mit grafischen Hügeln, Büschen und Wolken (5), sowie statische Hindernisse (Röhre) (6) sind fixe Gamebits. Die interaktiven Fragezeichen-, und Mauerblöcke (7) bilden persönlich gesehen eine Mischform. Sie sind nur unter bestimmten Bedingungen manipulierbar und beweglich und wechseln ihre Eigenschaft nach der Interaktion mit dem Spieler teilweise von variablen zu fixen. Beispielsweise werden die leicht beweglichen Fragezeichen nach dem Auslösen durch Mario zu leeren, fixen Hindernissen (8).

1.2 Definition - World Building

World Building: Das Erdenken, Designen, Kreieren von fiktionalen Welten/Universen und deren Inhalten (Geschichte, Kreaturen, Physik, etc.)

Dieser Term ist essentiell, da die praktische Arbeit zu dieser Thesis den Prozess des World Buildings veranschaulicht und durch selbst entwickelte Methoden, die in Kapitel 5 im Detail beschrieben werden, zu erforschen versucht.

World Building, oder zu Deutsch; „Welten erbauen/konstruieren“ ist ein Term, der in der aktuellen Spieleindustrie für das Bilden von fiktiven Welten oder sogar ganzer Universen, sowie all ihren Facetten, verwendet wird.

[13] Rehfeld, Gunter, Schmidt, Ulrich (Hrsg.): Game Design und Produktion. München. Hanser Verlag (2014) S.103

[14] Ebd. S.103

Dazu gehört nicht nur die Entwicklung einer visuellen Repräsentation dieser Welt, sondern auch das Schreiben einer passenden Geschichte (Story), die Entwicklung von Charakteren und Bewohnern dieser Welten, usw. In Literatur und Film werden ebenso ganze Universen erschaffen. *Der Herr der Ringe*[15] oder das *Marvel*-Comic Universum[16] mit den dazugehörigen Verfilmungen sind nur zwei Beispiele dafür. Der Begriff „world building" wurde populär, als ihn der britische Astrophysiker Arthur Stanley Eddington in seinem Werk „Space, time and gravitation: an outline of the general relativity theory" 1920 verwendete.[17] Eddington nutzte den Term sinngemäss um Gedanken und mathematische Formeln zu hypothetischen Welten mit anderen physikalischen Eigenschaften zu Beschreiben. Hier beispielsweise eine grobe Beschreibung Eddington's eines Universums ohne Gravitation:

> "space-time is curved in the second degree. This is the state of the world in a region not containing matter or electrons (bound energy), but containing light or electromagnetic fields (free energy)." [18]

So gesehen kann World Builing auch im Zusammenhang mit dem Science-Fiction-Begriff „Terraforming", also dem Aufbereiten von Landschaften oder ganzer Planeten zur zukünftigen Besiedelung durch Menschen, genutzt werden. Im Brettspiel *Terraforming Mars* [5] bildet das wirtschaften mit Mars-Ressourcen das Spielprinzip.

> „Jeder Spieler verkörpert dabei einen mächtigen Konzern, der seinen Teil zum Fortschritt beitragen und natürlich auch möglichst viel Gewinn abschöpfen möchte. Eine Partie endet, wenn es die Spieler geschafft haben, den Sauerstoffgehalt in der Mars-Atmosphäre auf 14 Prozent zu bringen, die Temperatur auf plus 8 Grad Celsius zu erhöhen und sich neun grosse Ozeane auf dem Planeten ausgebreitet haben. Am generellen Umbau des Mars arbeiten zwar alle gemeinsam, spielen aber sonst unerbittlich gegeneinander." [19]

Bild 5: Das Brettspiel Terraforming Mars für einen bis fünf Spieler

[15] Tolkien, John Ronald Reuel: The Lord of the Rings. London: Allen & Unwill (1954)

[16] (Gründer) Goodman, Martin. (Heute in Besitz von The Walt Disney Company): Marvel Entertainment, LLC (1939 mit Namen "Timely Comics")

[17] Eddington, Arthur Stanley: Space, time and gravitation : an outline of the general relativity theory. Cambridge: Cambridge Unitversity Press (1920)

[18] Eddington, Arthur Stanley: Space, time and gravitation : an outline of the general relativity theory. Cambridge: Cambridge Unitversity Press (1920) S.82

[19] Felber, Tom: Die Zähmung des Roten Planeten. Zürich. Neue Zürcher Zeitung (2017) https://www.nzz.ch/gesellschaft/spiel/spiele-kritik-zu-terraforming-mars-die-zaehmung-des-roten-planeten-ld.1298888 (27.05.2018)

1.3 Definition - Open-World-Game

Open-World-Game: Games mit offener, meist von Spielbeginn an frei begehbarer, nahtloser Spielewelt. Das Fortschreiten im Spiel ist dabei in den meisten Fällen nicht linear, sondern lässt den Spieler selbst entscheiden, wann er welche Aufgabe angeht.

Der Fokus auf prozedurale Generierung im praktischen Teil der Arbeit ist besonders im Genre der Open-World-Games spannend und vielversprechend einsetzbar. Um die Kriterien für solche Spiele zu kennen ist auch hier eine Definition angebracht.[20]

Open-World-Games weisen eine Reihe an Charakteristika auf, auf die bei der Nutzung des Begriffes Bezug genommen wird. Diese Charakteristika lauten wie folgt:

1. Offene Spielwelt, die zumindest in Teilen bereits von Beginn an erforscht werden kann
2. Aufgaben haben keine feste Reihenfolge und werden im Spiel in drei Kategorien aufgeteilt:
    a. Primäre Aufgaben – Haben direkten Einfluss auf den Fortschritt in der Spielgeschichte (Story) -> Für Erreichen des Spielziels unbedingt nötig
    b. Sekundäre Aufgaben (engl. Side Quests) – Haben indirekten Einfluss auf die Story, die das Fortschreiten erleichtern (Bsp. Geld sammeln, Waffen verstärken) -> Für Erreichen des Spielziels optional
    c. Tertiäre Aufgaben – Haben keinen unmittelbaren Storyeinfluss. Oft sind dies Minispiele, die den Spielspass erhöhen -> Für Erreichen des Spielziels optional
3. In vielen Open-World-Games gibt es optionale Fortbewegungsmitteln oder Portale (Schnellreisen), mit denen wortwörtlich schnelleres Reisen in der riesigen Spielewelt ermöglicht wird. Der Spieler soll sich bei der Fortbewegung in den massiven Welten nicht bei stumpfem Herumlaufen langweilen.

Beispiele für Games des Genre Open-World, auf die partiell in diesem Kapitel noch Bezug genommen wird, sind *Red Dead Redemption* [6] und *The Legend of Zelda: Breath of the Wild* [7].

Bild 6: Die weite Landschaft der fiktiven Region Diez Coronas im Territorium Nuovo Paradiso des Wild West Games Red Dead Redemption

[20] Da keine einheitliche und allgemeingültige Definition für das Spielgenre der Open-World-Games existiert, bedient sich diese Definition bei freien Wikis, die von Spielern und Spieleentwicklern, als kompetenteste Autorengruppe für diese Thematik, geschrieben werden. Zu erwähnen ist, dass bei diesem Vorgehen nur die für dieses Projekt als qualitativ genug und relevant empfundenen Definitionen übernommen wurden.

Bild 7: Das Schlachthaus (El Matadero) in Diez Coronas mit zwei Cowboys

Bild 8: Der Held Link blickt auf die riesige Welt von Hyrule. Der Spieler kann bereits zu Beginn weite Teile der Welt erkunden, wobei die Fähigkeiten von Link in frühen Stadien meist nicht ausreichen, um die gestellten Aufgaben zu lösen

Bild 9: Da die Welt bei diesem Game so gross ist, wurde unter anderem die Möglichkeit mit einem Gleiter zu fliegen oder wilde Tiere als Reittiere zu nutzen zur Beschleunigung des Reisetempos zum Spiel hinzugefügt

Eine Unterkategorie der Open-World-Games wird als „sandboxed“ oder „Sandkasten-Games“ bezeichnet. Ähnlich eines spielenden Kindes in einem gewöhnlichen Sandkasten, kann der Spieler hier in geregelter aber sehr intensiver Art auf die Spielewelt und Umgebung einwirken. Beispielsweise können wie in *Minecraft* [8] Teile der Welt abgebaut und als Ressource für eigene bauliche Interventionen oder als Verbrauchsobjekte (Nahrung, Heilung, etc.) genutzt werden.

Bild 10: Die Welt in Minecraft besteht aus Blöcken mit jeweils eigenen Eigenschaften und Ressourcen

Bild 11: Die Ressourcen der Blöcke können mit einer Spitzhacke (rechts unten im Bild) abgebaut werden. Diese gewonnenen Ressourcen werden dem Inventar (unten mittig) hinzugefügt und dienen der Erstellung neuer Objekte oder Bauwerke

Dieses Genre bietet besonders Explorern[21], also Freunden von Spielen, in denen weite Teile der Welt erkundet und erforscht werden können, grossen Spielspass.

1.4 Definition - Prozedurale Generierung in Games

Prozedurale Generierung (in Games): Von einem Computer selbst erstellte Spielinhalte auf Basis eines, in erster Instanz von Menschen programmierten Regelwerks.

Ein grosser Teil des Wissens in dieser Arbeit stammt aus einer Sammlung online frei zugänglicher, wissenschaftlicher Abhandlungen der Autoren Julian Togelius, Noor Shaker und Mark J. Nielsen zum Thema „Procedural Content Generation".[22] In der Einleitung ihrer Analyse definieren sie prozedurale Generierung folgendermassen:

> „The definition we will use is that Procedural Content Generation (PCG) is the algorithmic creation of game content with limited or indirect user input.[23] In other words, PCG refers to computer software that can create game content on its own, or together with one or many human players or designers. A key term here is "content". In our definition, content is most of what is contained in a game: levels, maps, game rules, textures, stories, items, quests, music, weapons, vehicles, characters, etc." [24]

Für den, in Kapitel 5 im Detail behandelten Shape Generator und des Prototypens eines teilweise prozedural erstellten Games in Kapitel 6, gilt besonderes Augenmerk der Generierung von zweidimensionalen Formen, die im Text als Shapes bezeichnet werden. Auf andere Aspekte wird im Kontext der prozeduralen Generierung in dieser Arbeit nur zum Teil (Storygenerierung unter Kapitel 2.3.2 und 6.5) oder gar nicht (Sound, etc.) eingegangen. Folgende Illustrationen beschreiben, wie ein Generierungsvorgang theoretisch und stark vereinfacht mit der Zuhilfenahme eines Ausgangswertes (Seed) ablaufen kann.

Ein sogenannter Seed (von engl. Samen) bildet den Startwert für die Generierung. Der Seed kann dabei verschiedene Formen haben. Beispiele für einen Seed sind:

Bild 12: Seed

- Ganze Zahlen: 1, 2, 24, 678 ...
- Dezimalzahlen: 0.1, 0.5, 7.934 ...
- Buchstaben: a, b, C, D ...
- Mehrere Buchstaben: abc, hallo, ThEsiS
- Daten/Uhrzeit: 30.05.2018, 23:15:30

---

[21] Richard Bartle unterteilt Spieler in vier Kategorien:
- Achiever: Möchten im Spiel alles erreichen (alle Punkte holen, das Spiel zu 100% schaffen)
- Explorer: Möchten die Spielwelt und ihre Details erkunden
- Socializer: Streben Onlinekontakte mit anderen Spielern an
- Killer: Suchen den Wettbewerb im Spiel (andere Spieler besiegen)

Diese Klassifizierung gilt heute als Massstab für Online-Spiele, unter der Annahme, dass kein Spieler in nur eine der Kategorien fällt. (Vergleiche: Rehfeld, Gunter, Schmidt, Ulrich (Hrsg.): Game Design und Produktion. München. Hanser Verlag (2014) S.15)

[22] Togelius Julian, Shaker Noor, Nielsen Mark J.: Procedural Content Generation in Games. A Textbook and Overview of current Research. Springer (2016). http://pcgbook.com (12.05.2018)

[23] Togelius,J.,Kastbjerg,E.,Schedl,D.,Yannakakis,G.N.:What is procedural content generation?. Mario on the borderline. In: Proceedings of the 2nd Workshop on Procedural Content Generation in Games (2011)

[24] Togelius Julian, Shaker Noor, Nielsen Mark J.: Procedural Content Generation in Games. A Textbook and Overview of current Research. Springer (2016) S. 1

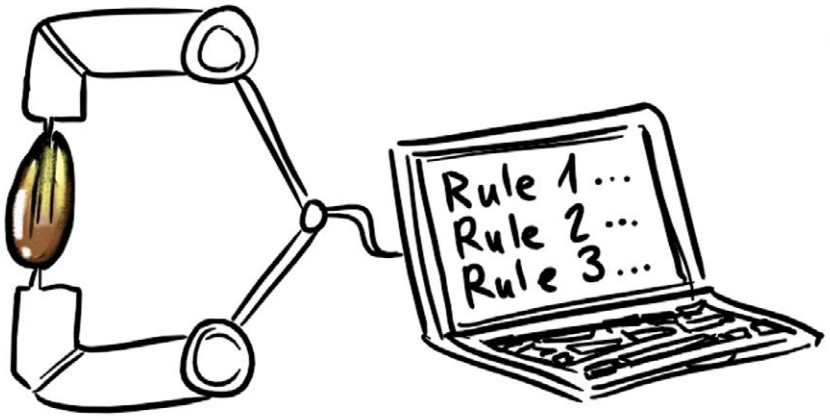


Bild 13: Computer nimmt Seed als Ausgangswert für die Berechnungen

Dieser Seed fungiert als Ausgangswert für ein Regelwerk, welches ein Mensch zuvor an einem Computer programmiert hat. Ein solches Regelwerk für ein generiertes Rechteck könnte wie folgt lauten:

Regel 1: Seed + Seed^2 = Breite des Rechtecks
Regel 2: (Seed - 1) + Seed^2 = Höhe des Rechtecks
Regel 3: falls Seed > 3 = Rechteck blau gefüllt

Nimmt man nun für den Seed einen Zahlenwert ergeben sich folgende Beispiel-Kalkulationen:

Seed = 2
2 + 4 = 6 (Breite des Rechtecks)
(2 - 1) + 4 = 5 (Höhe des Rechtecks)
2 > 3 trifft nicht zu (nicht blau gefüllt)

Rechteck A (Einheit Millimeter):

Seed = 5
5 + 25 = 30 (Breite des Rechtecks)
(5 - 1) + 25= 21 (Höhe des Recht-ecks)
5 > 3 trifft zu (Blau gefüllt)

Rechteck B (Einheit Millimeter):

Wie beim Rechteck können also Inhalte vom Computer selbstständig ausgegeben werden. Solche Inhalte können, je nach einprogrammiertem Regelwerk, sehr komplexe Formen annehmen. Nebst der visuellen Form von Objekten, lassen sich auch Geschichten (Stories) und Eigenschaften von Spielinhalten auf dieselbe Weise prozedural generieren. Mehr dazu unter Kapitel 2.3.2.

1.4.1 Für diese Arbeit ausschlaggebende Unterkategorien der prozeduralen Generierung

Im Rahmen dieser Arbeit sind folgende, selbstdefinierte Unterkategorien der komplexen Thematik um prozedurale Generierung ausschlaggebend, welche zur verbesserten Lesbarkeit in Englisch und in Kürzeln verwendet werden:

1. Non-agent-based Procedural Generation (no-agent Proc.Gen)
2. Artificial-Intelligence-agent-based Procedural Generation (AI-agent Proc.Gen)
3. Human-agent-based Procedural Generation (human-agent Proc.Gen)

Eine tiefergehende Ausführung über andere mögliche Untertypen können aus dem, in Kapitel 1.4 besprochenen Paper[25] entnommen werden. Als Agent wird bei den Bezeichnungen die aktiv eingreifende oder evaluierende Instanz genannt. Egal ob diese nun ein Computerprogramm oder ein Mensch ist.

1.4.1.1 Non-agent-based Procedural Generation (no-agent Proc.Gen)

Bei no-agent Proc.Gen wird das direkte Ergebnis des Generierungsvorgangs als vollwertiger Spielinhalt akzeptiert. Es folgen keine weiteren Evaluationsschritte, die die Inhalte auf Funktionalität oder auf ästhetische Kriterien hin bewerten. Diese Art der prozeduralen Generierung bildet zudem ein Teilstück der AI-, und human-agent Proc.Gen.

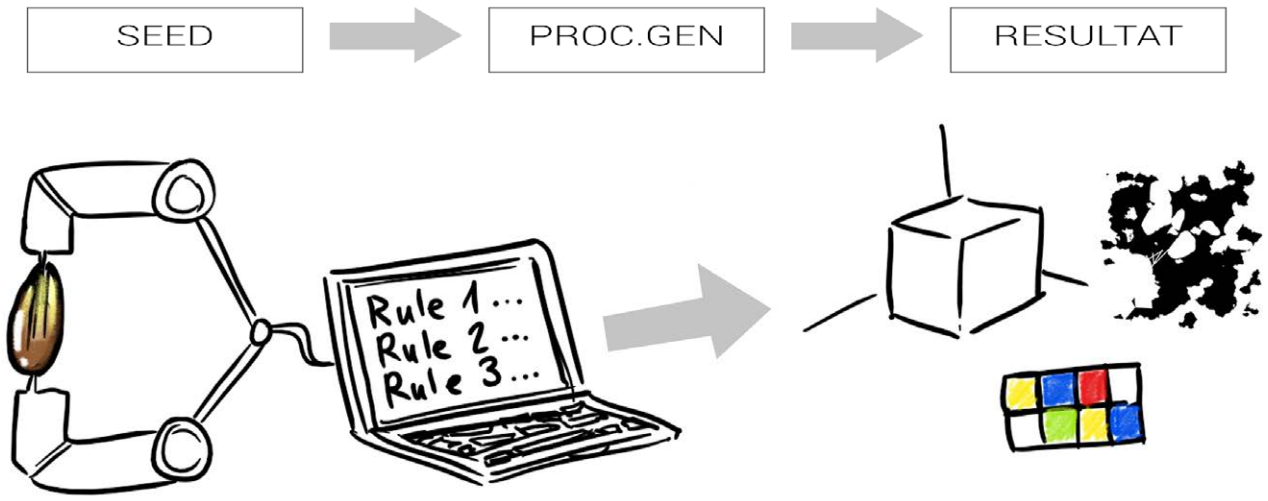


Bild 14: Seed als Ausgangslage für Proc.Gen mit verschiedenen Outputs

[25] Togelius Julian, Shaker Noor, Nielsen Mark J.: Procedural Content Generation in Games. A Textbook and Overview of current Research. Springer (2016). http://pcgbook.com (12.05.2018

1.4.1.2 AI-agent-based Procedural Generation (AI-agent Proc.Gen)

Bei der AI-agent Proc.Gen (AI = engl. Artificial Intelligence, zu Deutsch „künstliche Intelligenz“) findet eine Evaluation des Ergebnisses durch die Software statt. Das geschieht in der Regel iterativ, also wiederholend (Generieren-Auswerten-Generieren-Auswerten) bis ein gültiges Resultat erreicht ist. Gültig bedeutet hier: gewissen einprogrammierten Kriterien (Werten) entsprechend. Die Resultatgewinnung kann auch evolutiv geführt werden. Das bedeutet, dass im Fall von mehreren möglichen Resultaten nur das, oder die Resultate einen nächsten Generationsprozess durchlaufen, die im Vergleich zueinander die besseren Werte aufweisen.

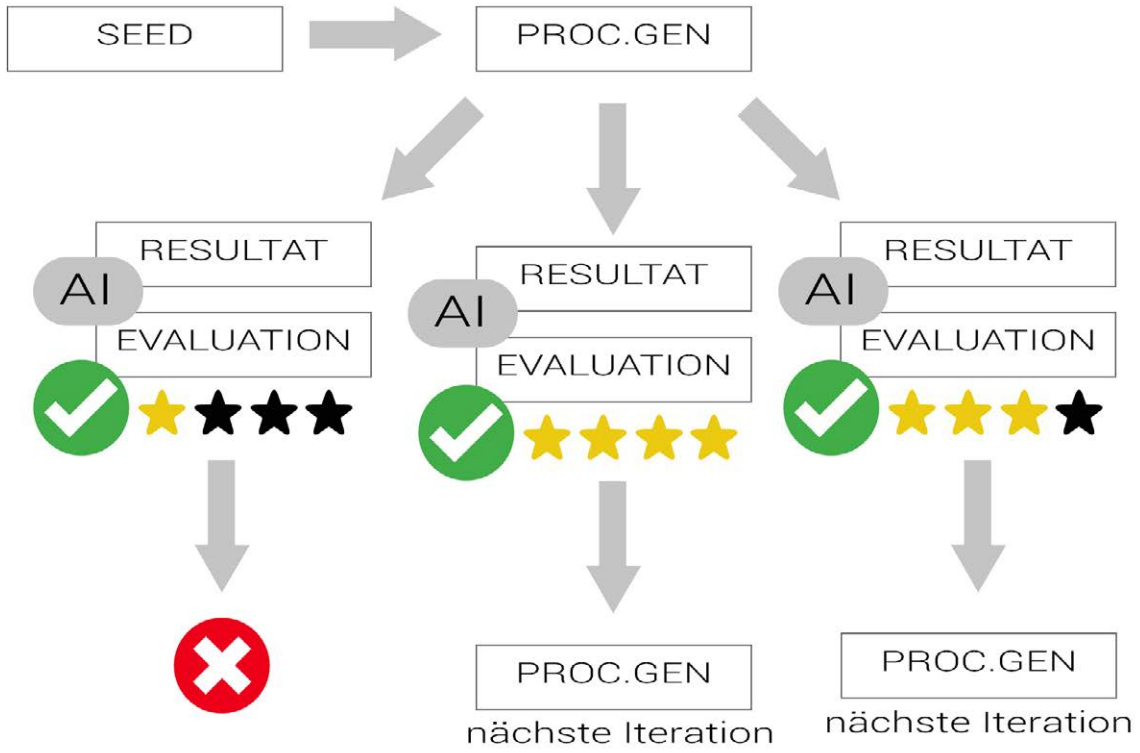


Bild 15: Schema AI-agent Proc.Gen

Dass es auch andere Ansätze, wie beispielsweise „beste 20 und schlechteste zwei Resultate kommen weiter“ gibt, ist klar. In der Kette der Generierung und Auswertung von AI-agent Proc.Gen bilden mehrere no-agent Proc.Gen die Glieder.

1.4.1.2 Human-agent-based Procedural Generation (human-agent Proc.Gen)

Bei der human-agent Proc.Gen agiert anstelle oder ergänzend zu AI-, bzw. no-agent Proc.Gen ein Mensch als Evaluationsinstanz. In der Regel ist diese Person ein Designer, der die Resultate verwenden, verwerfen oder auf diversen Wegen weiter manipulieren kann. Diese Art der prozeduralen Generierung ist für diese Arbeit besonders unter Kapitel 6, der Erstellung des Prototyps, von essentieller Bedeutung.

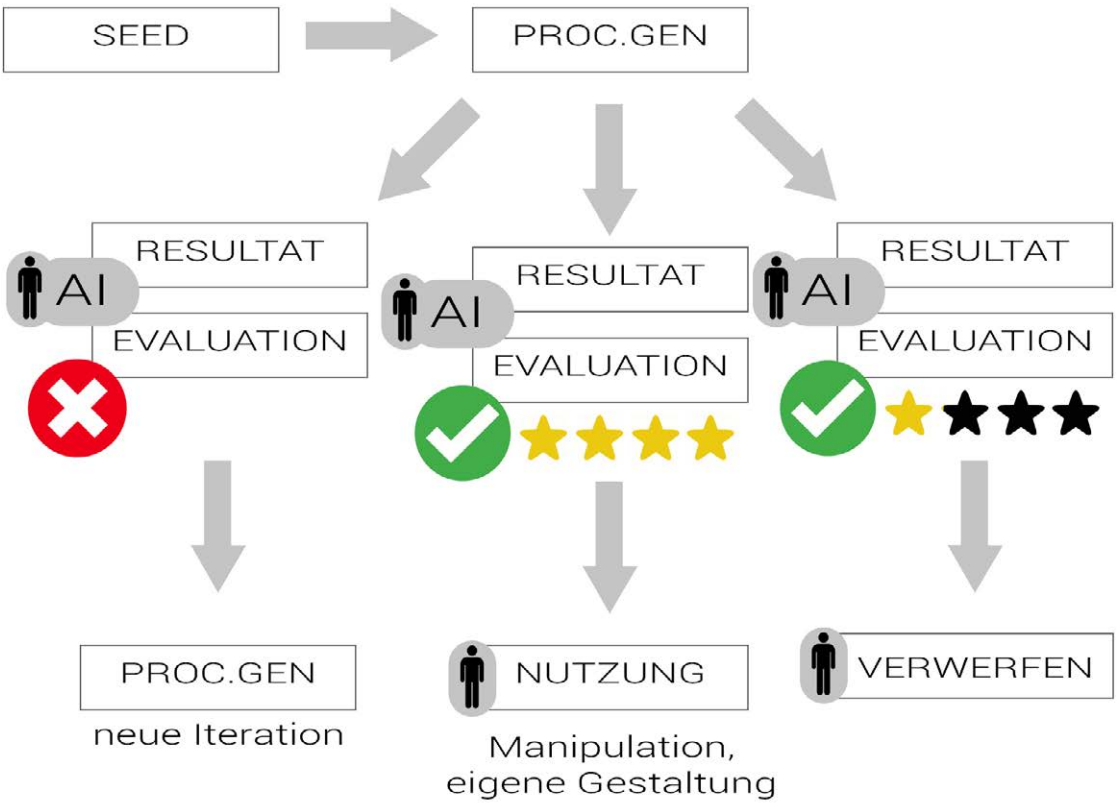


Bild 16: Schema human-agent Proc.Gen

1.5 Definition - „Elektronisch-generatives“ Design

Da als „generatives Design“ auch die Gestaltung ausserhalb des Computerkontextes verstanden werden kann, ist es sinnvoll auch hier eine kurze Definition anzubringen. „Generativ“ bedeutet „erzeugend“ oder auch „hervorbringend“ und wird als Adjektiv in den verschiedensten Disziplinen der Technik, des Designs oder der Kunst verwendet.[26] Design wird in dieser Arbeit als jegliche Form des Gestaltens verstanden und kann sowohl von Menschen, allgemein „Lebewesen“, der Natur oder Software ausgeführt werden. Als verdeutlichendes Gegenstück zum elektronisch-generativen Design, also dem „nicht-elektronisch-generativen Design“ könnten beispielsweise die Strukturen auf Sanddünen in einer realen Wüste gelten. Diese Wellenstrukturen werden in der Regel von unbeeinflussbaren Komponenten (Wind, Haftreibung der Sandkörner, etc.) generiert, bzw. designt.

Bild 17: Nicht-elektronisches-, oder „natürlich-generatives Design“. Die durch Umwelteinflüsse entstehenden Strukturen auf Sanddünen

[26] Wikipedia. Begriff: Generativ https://de.wikipedia.org/wiki/Generativ (13.05.2018)

Beim elektronisch generativen Design spielt sich der Prozess der Generierung in digitaler Form auf elektronischen Geräten, wie bspw. Computern ab. Die spezifisch dafür angelegte Software, die auf dem Gerät ausgeführt wird, bringt somit elektronisch generierte Inhalte hervor.

Die unter Kapitel 6.1 vorkommenden Heightmaps, also Bilddateien, die wie Landkarten zur Bestimmung von Höhenunterschieden verwendet werden, sind Beispiele für (prozedural) elektronisch generierte Bilder.

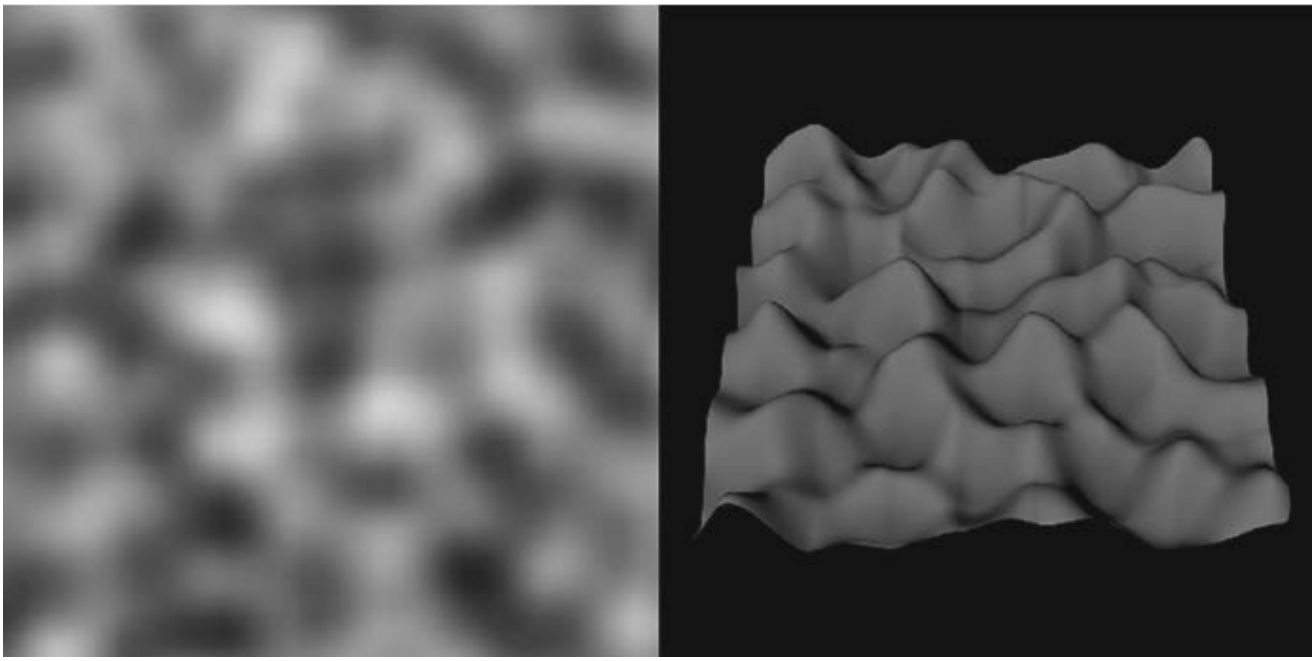


Bild 18: Links: Prozedurale 2D-Heightmap. Rechts: Generiertes Terrain aus 2D-Heightmap

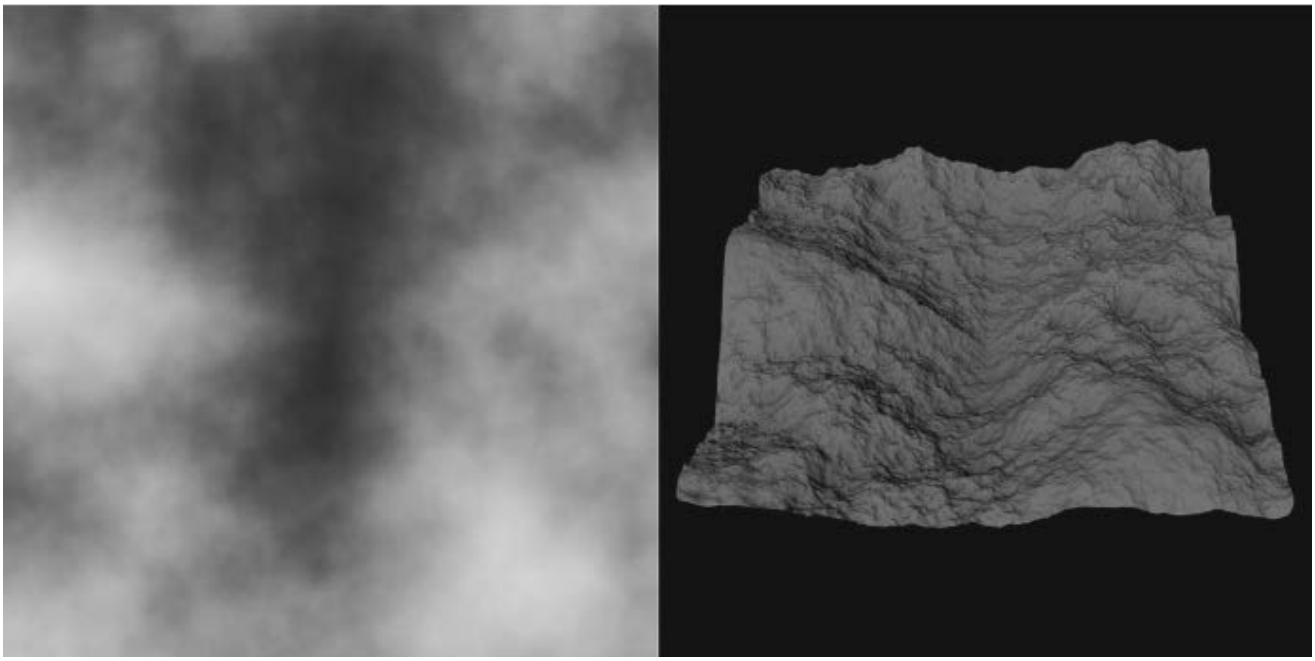


Bild 19: Selbes Schema wie bei Bild 18 mit einer höheren Detailstufe

1.6 Definition - Bildsprache

In dieser Arbeit kann „Bildsprache“ in seiner Bedeutung mit „Stil“ gleichgesetzt werden. Eine einheitliche und in sich geschlossene Bildsprache zu finden, ist in der Videospielindustrie durchaus eines der bedeutendsten Kriterien, wenn es um die visuelle Umsetzung der Inhalte geht. Die zur Verfügung stehenden, technischen Möglichkeiten können sich dabei stark auf die Bildsprache des Games auswirken, wie es beispielsweise bei der Entwicklung der weltweit bekannten *Super Mario* Spielereihe der Fall war. Shigeru Miyamoto, der Erfinder der Videospielfigur Mario, verwendete wegen Zeitdrucks und der limitierten Speicherkapazität der damaligen Speichermedien für das Spiel *Super Mario Bros.* Bildelemente der Welt mehrfach wieder, was der visuellen Einheit des Spiels zu Gute kam.

> „Developers were short on space due to the limited nature of the NES cartridges and recycling assets was a great way to save space.“ [27]

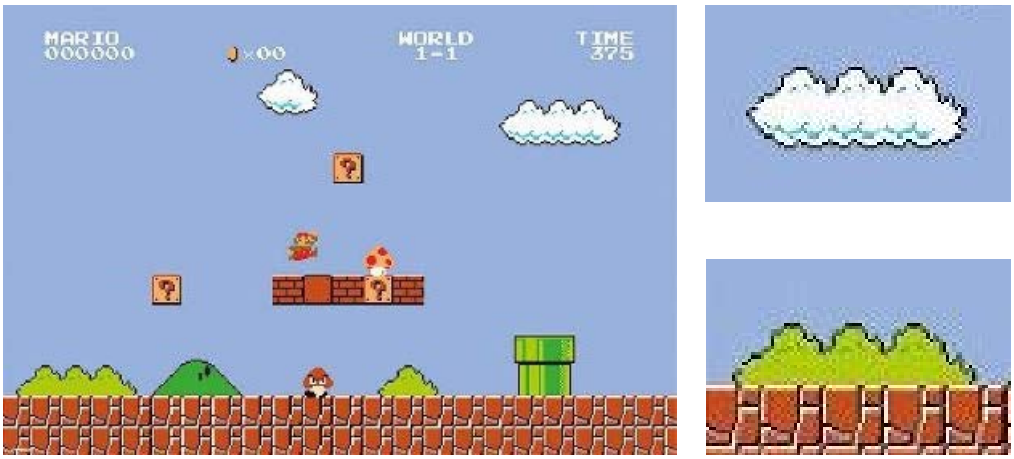


Bild 20: Erstes Level von Super Mario Bros. mit wiederkehrenden Designelementen (Wolke = Busch)

Aus stilistischen Gründen und aus dem Streben nach einem hohen Grad an visueller Kohärenz geht in dieser Arbeit hervor, dass Stilbruch (das Nicht-Zusammenpassen von visuellen Elementen innerhalb des Gesamtdesigns) negativ bewertet wird. Dies ist hinsichtlich der Entwicklung des Prototyps *Geno-Terra* in Kapitel 6 von grosser Bedeutung.

1.6.1 Formale Gestaltungsprinzipien der Bildsprache

Das folgende Schema der Kunstvermittler Stéphanie Lobmaier und Thomas Schatz dient zur Verdeutlichung einiger bildsprachedefinierender Prinzipien, welche hier unter dem Begriff der Bildsprache zusammengefasst werden.

> „[Die formalen Gestaltungsprinzipien] lassen sich in verschiedenen gestalterischen Bereichen anwenden, wie unter anderem im Layout des Grafikdesigns, in der Bildkomposition einer Zeichnung oder bei der Inszenierung mit Fotografie.“ [28]

---

[27] http://www.levelbased.com/database/trivia/2-super-mario-bros-cloud/ (13.05.2018)

[28] Lobmaier, Stéphanie. Schatz, Thomas: Kleines ABC der Bildsprache. Formale Gestaltungsprinzipien der Bildsprache, Grammatik der Bildsprache. (2017) http://kunstunterricht.ch/cms/grundlagen/224-grammatik-der-bildsprache-formale-gestaltungsprinzipien (13.05.2018

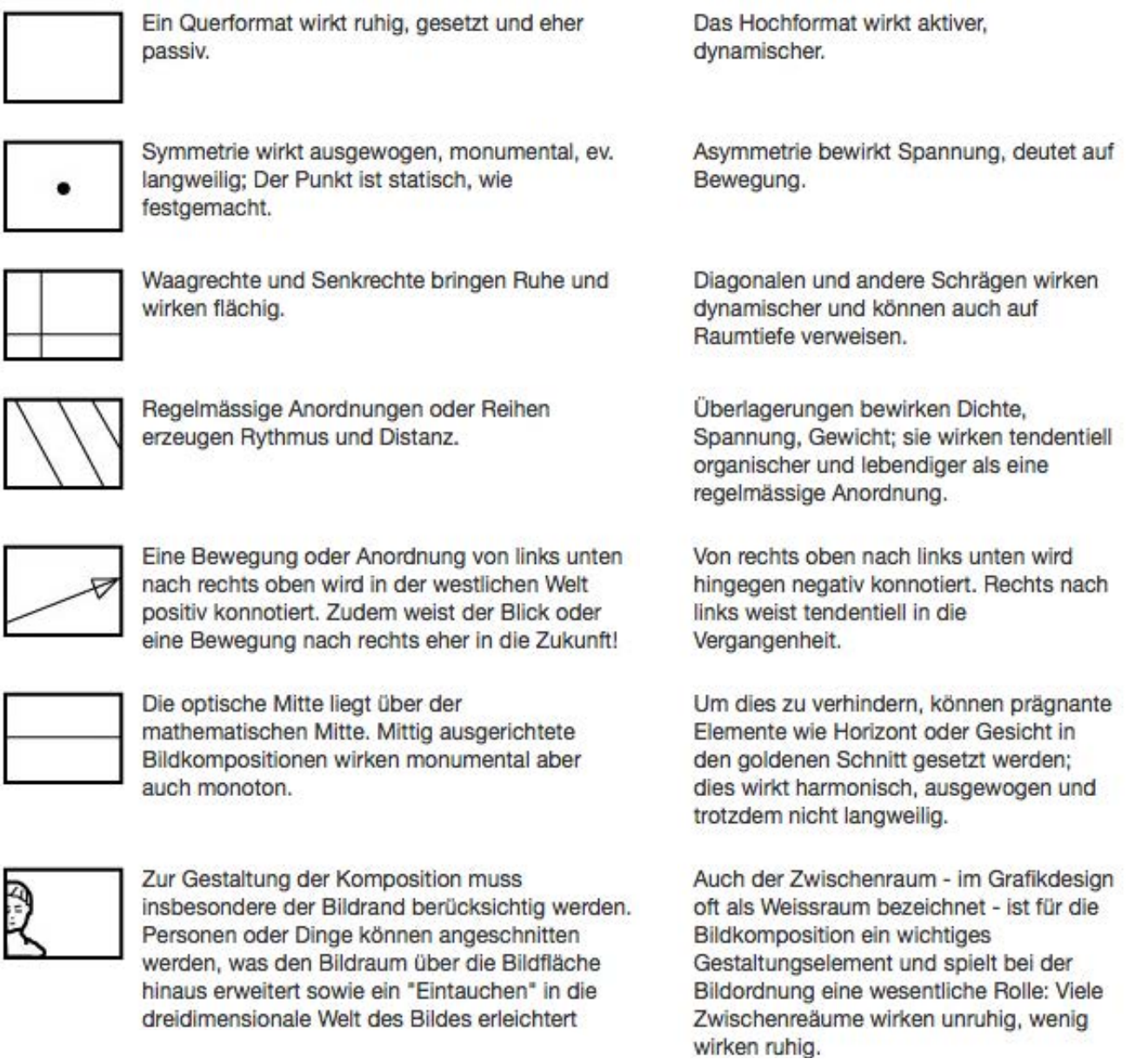


Bild 21: Schema formaler Gestaltungskriterien (Screenshot)

Games als interaktives Bewegtbildmedium werden in der Auflistung nicht konkret genannt. Im Rahmen dieser Auseinandersetzung wird das Schema als gültig erklärt, da auch Gamewelten, trotz ihrer scheinbaren Dreidimensionalität (bei 3D-Games), hauptsächlich über zweidimensionale Bildschirme erfahrbar sind. In Kapitel 1.6.2 wird deshalb zur Vervollständigung auf die Rolle der Bildsprache im Bewegtbild verwiesen

1.6.2 Ausarbeitung des Kohärenzbegriffes und dessen Kriterien für diese Arbeit

Kohärenz: Zusammenhang der Einzelteile zu einem einheitlichen Ganzen

Das Thema der Kohärenz, zu welchem in Kapitel 3 weitere Beispiele aus Literatur und Film zu finden sind, spielt in der theoretischen Auseinandersetzung dieser Arbeit die wesentlichste Rolle. Der Begriff „Kohärenz" stammt aus dem lateinischen und bedeutet in etwa „Zusammenhang".[29] Kohärenz der Bildsprache wird in dieser Arbeit somit als zusammenhängende (einheitliche) Gestaltung, kurz: „stilisiertes Design" verstanden. Eine visuell nicht kohärente, also nicht einheitlich gestaltete Welt, führt in der Regel zu einer ästhetisch weniger anregenden, visuellen Erfahrung des Games (vergleiche Kapitel 1.6). Parallelen zur bildenden Kunst können hier begrifflich gezogen werden. Auch die Epochen der bildenden Kunst sind untereinander stilistisch in der Regel gut abgrenzbar.

Bild 22: Caspar David Friedrichs „Böhmische Landschaft" um 1810/11

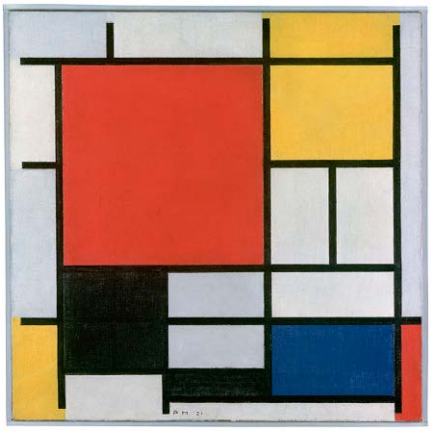


Bild 23: Piet Mondrian's „Komposition mit Rot, Gelb, Blau und Schwarz" um 1921

Die Entstehungsdaten der beiden gezeigten Werke liegen über 100 Jahre auseinander und sind visuell betrachtet praktisch gar nicht miteinander verwandt. Dennoch wollen beide Künstler mit ihrer persönlichen Stilisierung ihre ganz eigene Vorstellung von „Welt" aufzeigen. In ihrer Erscheinung sind beide Gemälde kohärent. Sie sind eine jeweilige, visuelle Einheit. Eine vollständige Analyse beider Werke würde den Rahmen dieser Auseinandersetzung sprengen, weshalb an dieser Stelle darauf verzichtet wird.

David O'Reilly, ein zeitgenössischer Animationsfilmemacher und Schöpfer des bereits in Kapitel 1.1.1 besprochenen digitalen Erlebnisses *Everything*, schreibt in seinem Essay Basic Animation Aesthetics[30] zum Schlagwort „Kohärenz":

> „My central belief is that the key to aesthetics is coherence. In 3d we essentially create artificial models of worlds, I contend that what makes these worlds believable is simply how coherent they are; how all the elements tie together under a set of rules which govern them consistently. This coherence spreads to all areas of a film; dialogue, design, sound, music, movement etc. Together they create a feedback-loop which reaffirms that what we are looking at is true. The human eye wants this aesthetic harmony." [31]

Diese Arbeit orientiert sich an der Annahme, dass ein hoher Grad an Kohärenz des visuellen Eindrucks eines Games, dessen Wertigkeit für den Spieler erhöht. Wenn man die Kohärenz werten oder den Grad an visueller Bildsprachen-Kohärenz messen möchte, wie es in der Evaluation des Prototyps unter Kapitel 7 der Fall ist, müssen zuerst Kriterien entwickelt werden.

---

[29] https://www.duden.de/rechtschreibung/Kohaerenz (13.05.2018)

[30] Downloadlink: http://www.davidoreilly.com/downloads/ (09.06.2018)

[31] O'Reilly, David: Basic Animation Aesthetics (2009) S.2

Dies bedingt nicht den Anspruch ein allgemeingültiges Regelwerk für kohärente Gestaltung von Videospielen zu proklamieren, denn wie O'Reilly schon für das Medium des Filmes schreibt und was für Games als verwandtes Medium hier ebenso übernommen wird:

> „In a film, the exact rules and limitations of its world can only ever truly be known to the filmmaker. All feelings, sounds, moods, characters, colors, shapes and so on should be immediately obvious to him or her. It's always a good idea to lay down the aesthetic rules which permeate the world you are creating." [32]

O'Reilly schliesst seinen Essay mit den Worten:

> „I feel we should forget everything about the idea of right or wrong, of beauty and ugliness, and focus on the idea of coherence." [33]

Diese Textausschnitte machen klar, dass der Designer des Games oder bei grossen Produktionen der Artdirector[34] der Spieleentwicklungsfirma, die Regeln der Gestaltung festlegt und stetig auf dessen visuelle Kohärenz überprüft. Nebst der visuellen Kohärenz muss ein Designer natürlich auch die Story des Spiels in seinen Entscheidungen mitberücksichtigen. Darauf wird in Kapitel 6.5 das Vorgehen exemplarisch am Prototyp erläutert. Generell wende ich in dieser Arbeit folgende Kriterien an:

- Designentscheidungen sind bindend und werden bei allen visuell erfahrbaren Inhalten des Games gleich angewendet
- Alle visuell erfahrbaren Inhalte gehen von einer Gestaltungsbasis aus, die aus derselben no-agent Proc.Gen mit einheitlichem Seed entstanden sind

Im Rahmen des Prototyps werden Inhalte als all das definiert, was Spieler in der Spielewelt antreffen oder sehen können. Die Musik zählt somit in diesem Projekt nicht zu jenen Kriterien, obwohl sie für den Prototyp spezifisch ausgewählt und angewandt wird. Eine konkrete Anwendungsform der genannten beiden Kriterien ist die, für diese Arbeit entwickelte ***Shape Based TFFT-Methode*** aus Kapitel 5.

1.7 Kontextualisierung von prozeduralem Design in der Spieleindustrie

Nachdem prozedurale Generierung definiert ist, stellt sich selbstverständlich die Frage in welchen Bereichen der Spieleentwicklung prozedurales Design heute angewendet wird und welche Vorteile eine Software haben kann, die sich dieses Prozesses bedient.

Der wohl bedeutendste und einleuchtendste Anwendungsbereich für prozedurale Generierung in der Spieleentwicklung ist überall dort, wo ein einprogrammiertes Regelwerk in Form eines oder mehrerer Algorithmen, in kürzerer Zeit oder sehr viel einfacher mehr funktionierende Resultate hervorbringen kann, als menschliche Designer bei manueller Arbeit.

---

[32] O'Reilly, David: Basic Animation Aesthetics (2009) S.2

[33] Ebd. S.2

[34] Der Artdirector (die künstlerische Leitung) begleitet, leitet und überwacht die künstlerische Umsetzung von Projekten. Besonders geläufig ist die Bezeichnung in der Film-, und Videospielbranche

beispielhaft dafür sind Projekte, bei denen Spieler dasselbe Spielprinzip auf unendlich viele Ausgangssituationen anwenden können. Nimmt man zum Beispiel einfache, zweidimensionale Labyrinthe, so könnte ein Programm geschrieben werden, welches prozedural unter der Regel (Labyrinthgrösse = X Mal Y Pixel, 1 Eingang, 1 Ausgang, 1 Weg von Eingang zu Ausgang, Seed = Anzahl Irrwege) theoretisch unendlich viele Labyrinthe erstellen kann.

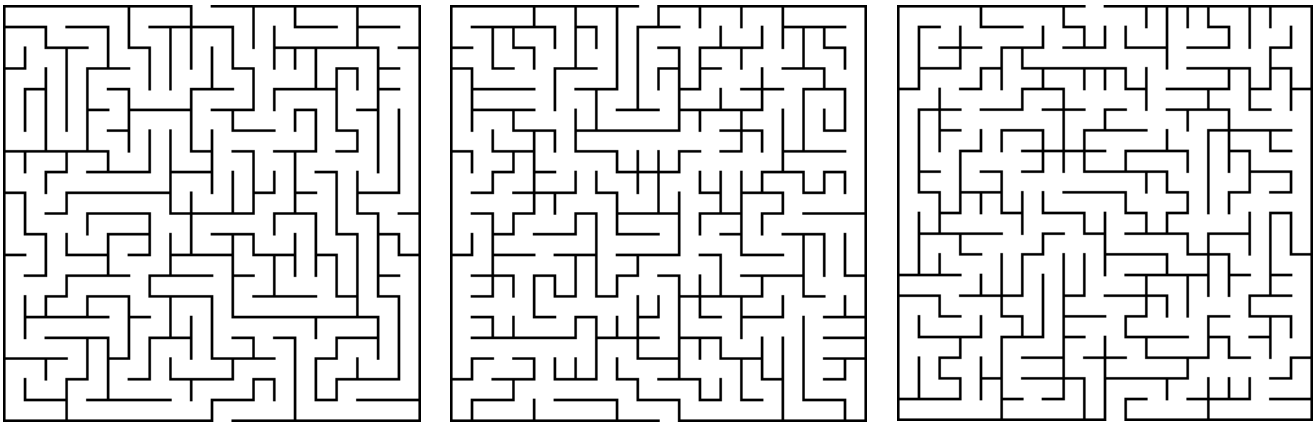


Bild 24: Drei Varianten eines generierten Labyrinthes mit verschiedenen Ausgangswerten

Solche „Labyrinthprinzipien" werden in vielen Dungeon-Spielen (Games bei welchen sich der Spieler durch ein Verlies bis zum Ziel durchkämpft) in weit komplexerer Form angewendet. Es gibt diverse Algorithmen, welche speziell für das Erstellen von Labyrinthen programmiert wurden.[35] Das Computerspielspiel *Rogue* [9] welches in Kapitel 2.3.1 als Beispiel für einfache Weltengenerierung dient oder auch *Diablo 3* [10], bei dem die Wege durch das Verlies teilweise prozedural erstellt werden, sind dafür exemplarisch. Auch andere Level-Generatoren laufen nach denselben Prinzipien ab. Wichtig im Diskurs über prozedurale Generierung ist die Tatsache, dass generisch wirkende Inhalte von den Designern möglichst vermieden werden wollen. Die wichtigen Räume der Dungeons in *Diablo 3*, in denen die grossen Monster (Boss-Monster) warten, werden nach wie vor manuell designt, um den Spieler einmalige Spielmomente und Eindrücke zu bescheren und somit den Spielspass zu erhöhen. Ein Mitarbeiter beim Projekt *Diablo 3* schreibt über die Dungeons im Spiel:

> „Our interiors are randomized, but we do some things differently that help make them more interesting. I think we accomplish this mostly by using our interior jigsaw pieces more intelligently, building more and different types of jigsaw pieces, and also because our artists are amazing." [36]

Bild 25: Links: Karte eines Dungeons in Diablo3. Rechts: Kampf gegen einen Endgegner (Boss) in Diablo3.

[35] Buck, Jamis: Maze Generation: Algorithm Recap (2011) http://weblog.jamisbuck.org/2011/2/7/maze-generation-algorithm-recap (30.05.2018)

[36] Username: Mjsk. Random Map Generation in Diablo 3 (2012) https://us.battle.net/forums/en/d3/topic/6147286293 (30.05.2018)

Das Platzieren von Unmengen an Vegetation in Open-World-Games, das als wichtiger Beitrag dieser Arbeit in Kapitel 6.2 zur Sprache kommt, geschieht zu grossen Teilen prozedural. Dafür ist konkret bereits Software wie *InfiniGrass*[37] zur Erweiterung von Funktionen von Spieleentwicklungsumgebungen wie Unity entwickelt worden.

Auch *Elite* [11] (siehe Kapitel 2.3.3.1) aus dem Jahr 1984 oder das erst 2017 erschienene Game *No Man's Sky* [12], das als „DAS prozedurale Spiel"[38] bekannt wurde, sind prozedural erstellte Games. Die Spielsoftware von *No Man's Sky* ist theoretisch sogar in der Lage ein Universum mit mehreren Trillionen Planeten inklusive eigener Flora und Fauna zu generieren.[39] Auch dies wäre ohne besagte Nutzung von prozeduraler Generierung nicht zu bewältigen, da die Speicherkapazität aller Rechner begrenzt ist.

> Prozedurale Generierung wird also dort genutzt, wo Inhalte nicht bereits zu Beginn des Games geladen werden müssen, sondern erst, wenn der Spieler betreffende Inhalte wahrnimmt oder mit ihnen interagiert. Dieses Vorgehen optimiert die Rechenleistung und regelt die computerinternen Ressourcen-Handhabung.

Ein weiteres konkretes Anwendungsfeld ist das Generieren von Spielelementen, die sehr weit vom Spieler, bzw. dessen Spielfigur (Avatar[40]) entfernt sind und bei denen visuelle Schönheitsfehler kaum auffallen.

Es existieren noch weitere Anwendungen, deren ausführliche Beschreibung und Analyse den Rahmen dieser schriftlichen Thesis sprengen würden und auf die ich daher nicht eingehen werde. Auch ist das Feld der prozeduralen Generierung in der Spieleentwicklung sehr dynamisch und wird mit der steigenden Automatisierung digitaler Abläufe an Bedeutung gewinnen.

1.8 Hypothese: „Generatives Design kann in Open-World-Games zur Erzeugung kohärenter Bildsprachen eingesetzt werden."

Im Rahmen dieser Arbeit wird folgende Hypothese auf ihre Validität überprüft:

Generatives Design ist eine neue Möglichkeit kohärente Gesamtbildsprachen in Open-World-Games zu erzeugen.

Zur Beantwortung, ob die Hypothese zutrifft, wird ein unter Kapitel 6 im Detail beleuchteter Prototyp erstellt und auf visuelle Kohärenz der Bildsprache hin untersucht. Dazu werden die Kriterien aus Kapitel 1.6.2 zur Auswertung dienen.

---

[37] Artengame: https://assetstore.unity.com/packages/tools/particles-effects/infinigrass-next-gen-interactive-volume-grass-45188 (14.08.2018)

[38] McKendrick, Innes: Continuous World Generation in No Man's Sky. Digitally Speaking (2018) https://www.gdcvault.com/play/1024265/Continuous-World-Generation-in-No (14.08.2018)

[39] Baker, Chris: 'No Man's Sky': How Games are building themselves (2016) https://www.rollingstone.com/culture/news/no-mans-sky-how-games-are-building-themselves-w433492 (14.05.2018)

[40] Vergleiche: Kapitel 1.1.2

1.9 Methode(n) – Entwicklung eines Game-Prototyps zur Untersuchung der Hypothese

Der Game-Prototyp wird bezüglich der visuellen Kohärenz seiner Bildsprache untersucht. Dazu werden die Kriterien aus Kapitel 1.6.2 zur Beurteilung dienen. Die Erfahrungen im Prozess der Spielegestaltung werden unter Einbezug der Rechercheergebnisse reflektiert.

Schliesslich wird ein sogenanntes Playtesting (Laien-Testpersonen spielen den Prototypen) durchgeführt. Die Interviews mit den Testspielerinnen/-spielern sollen einen Hinweis auf den Erfolg oder Misserfolg der visuellen Kohärenz geben (Wirkungsuntersuchung). Mit Blick auf die Produktionsmethode (Integration generatives Design in die Spielegestaltung) wird die in Kapitel 5 eingeführte ***Shape Based TFFT-Methode*** beim Erstellen des Prototyps bezüglich ihrer Brauchbarkeit in diesem spezifischen Projekt begutachtet.

2. Manueller Designprozess versus prozeduraler Designprozess

Dass die prozedurale Generierung ein Thema von hoher Bedeutung für die Spieleindustrie ist, wurde in Kapitel 1.7 verdeutlicht. In folgenden Kapiteln werden nun Unterschiede zwischen prozeduralen und manuellen Gestaltungsprozessen und eine Auswahl konkreter Anwendungsbeispiele aufgezeigt.

> Als manueller Designprozess ist hier, das von menschlichen Designern[41], unter Zuhilfenahme von Software, vollzogene Gestalten gemeint. Die Software sollte im Optimalfall für die Definition keine der prozeduralen Prozesse aus Kapitel 1.4.1 aufweisen.

2.1 Status Quo: Manuelles Spielweltendesign in Games

Da ein methodischer Abgleich diverser Designmethoden im Spieleweltdesign wegen der schieren Vielfalt und persönlicher Präferenzen der Entwickler (Studios) im Rahmen der Thesis nicht sinnvoll ist, reflektiere ich hier primär meine persönlichen Erfahrungen in der Arbeit mit der Unity-Engine[42] im Kontext des Spieleweltdesigns.

> **Unity:** Spieleentwicklung-Software für Privatpersonen und Firmen/Studios.

Weil der Prototyp zu dieser Arbeit sich besonders stark auf das Gestalten von Umwelt und Lebewesen im Spiel fokussiert, sind diese Punkte folgend genauer beschrieben. Der manuelle Prozess beim Gestalten von Umwelten oder ganzer Szenerien in Games ist komplex und beinhaltet verschiedenste, projektabhängige Teilbereiche mit dafür ausgebildeten Fachkräften. Im folgend eingeschobenen Kapitel wird ein Blick auf die geläufigen Designmöglichkeiten geworfen.

---

[41] „Das Konzept des generativen Designs, des Maschine-Learning und der KI legt nah, dass es nicht-menschliche Designer geben könnte – ein Thema, das für die Weiterentwicklung und die ethische/humanistische Betrachtung der Disziplin immanent wichtig sein wird." Ralf Michel (2018)

[42] https://unity3d.com (15.05.2018)

2.1.1 Manueller Designprozess von Umwelten (Environment-Design) in der Spieleentwicklungsumgebung Unity

Im Rahmen dieser Arbeit wird auf das manuelle Designen von Landschaftsformen und das Platzieren von Vegetation und anderen statischen Objekten in der Spielwelt Rücksicht genommen, während das Licht-Design (Lighting) und die Musik nicht behandelt werden. Auch wird der Prozess aus der Sicht einer einzelnen Privatperson beschrieben und nicht im Kontext einer multidisziplinären Spieleproduktion mit Art Direktoren.

Das Designen von Landschaftsformen kann in Unity mit dessen eingegliederten Terrainsystem angegangen werden.

> **Terrain:** Gelände, Topografie einer Landschaft

Dieser Vorgang wird wegen der einfachen Verständlichkeit exemplarisch für manuelles Umweltdesign in dieser Arbeit dienen. Im Terrainsystem wird eine zunächst flach ausgelegte 3D-Modell-Landschaft, mit digitalen Pinseln (Brushes) mit dem Mauscursor, manuell verändert. Berge und Hügel lassen sich so gut kontrolliert auf die gewählte Stelle der 3D-Landschaft auftragen oder abtragen.

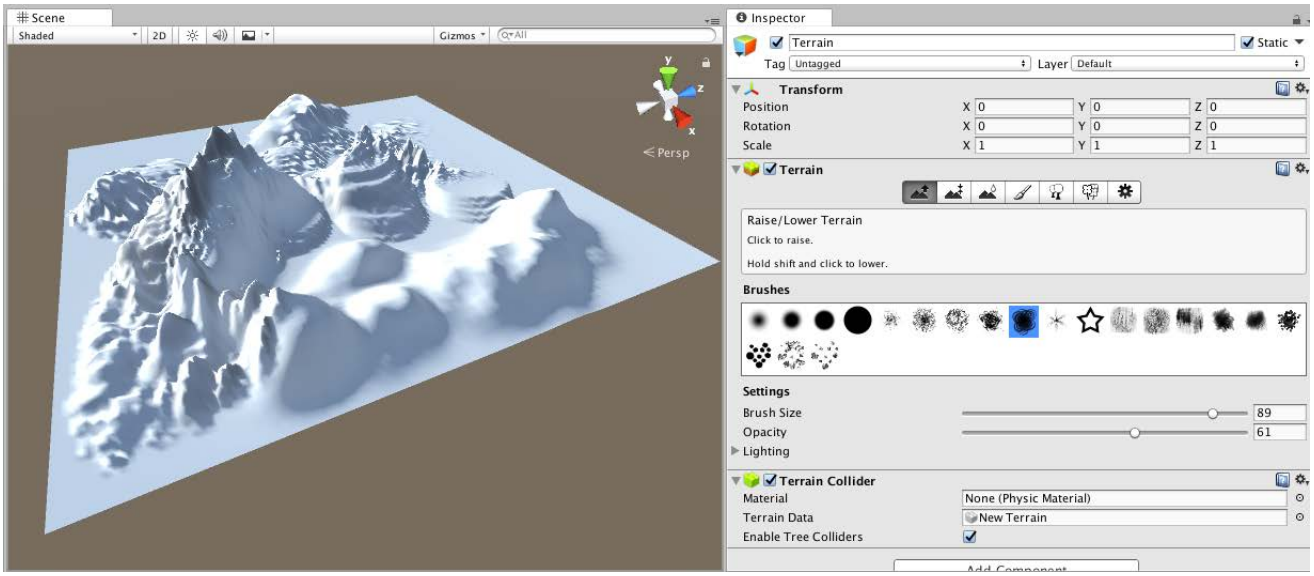


Bild 26: Manuelles Modellieren eines Terrains in Unity

Die Textur der Landschaft kann mit derselben Technik, also mithilfe der Maus mit dem digitalen Pinsel, aufgetragen werden. Dabei wird ein zuvor definiertes quadratisches Bild (Textur) auf die 3D-Oberfläche der Landschaft projiziert, dessen Aussehen verändert und als sogenanntes „Material" der 3D-Landschaft gespeichert.

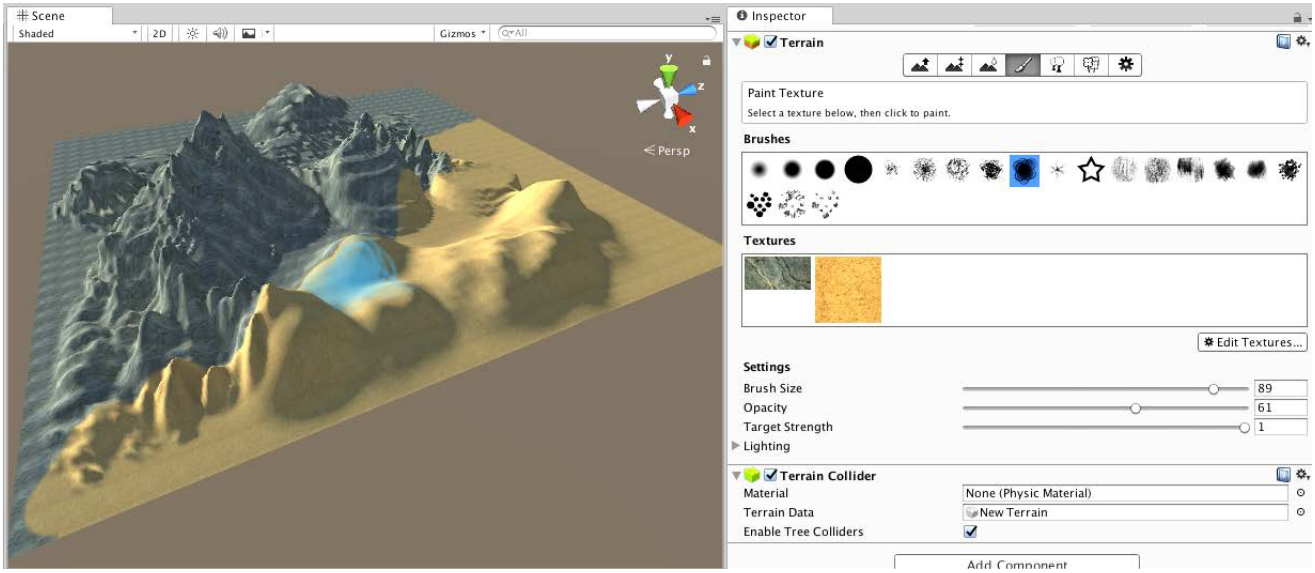


Bild 27: Manuelles Texturieren des Terrains mit Brushes

Auch das Platzieren von Vegetation oder anderen zunächst statischen 3D-Objekten läuft für die grobe Auslegung der Landschaftsdetails nach demselben Prinzip ab. Der Designer wählt die Objekte, die er auf der Landschaft erzeugen will, und nutzt den Pinsel und dessen Einstellungen (Dichte, Variation, etc.) für die Platzierung. Hier gilt es zu beachten, dass einige dieser Prozesse mit hoher Wahrscheinlichkeit bereits prozedurale Merkmale aus Kapitel 1.4.1, insbesondere des no-agent Proc.Gen aufweisen.

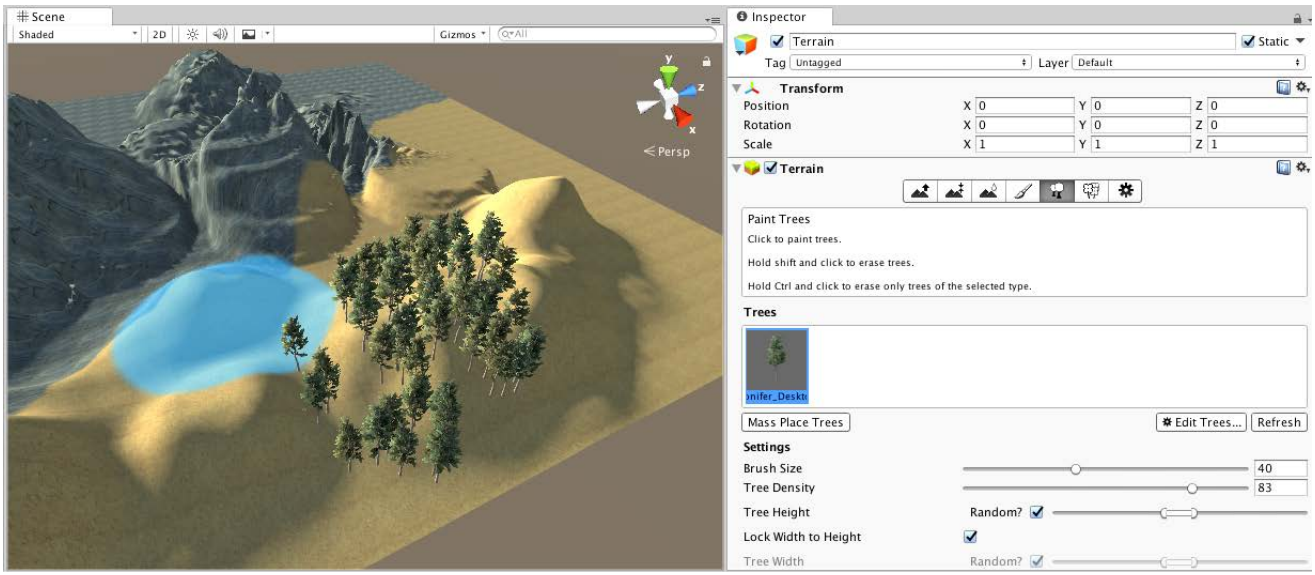


Bild 28: Manuelles Platzieren der Vegetation (hier: 3D-Modelle von Bäumen)

Die Funktion „Mass Place Trees“ ermöglicht gar ein vollautomatisches, jedoch grobes Platzieren der Vegetation auf dem gesamten Terrain.

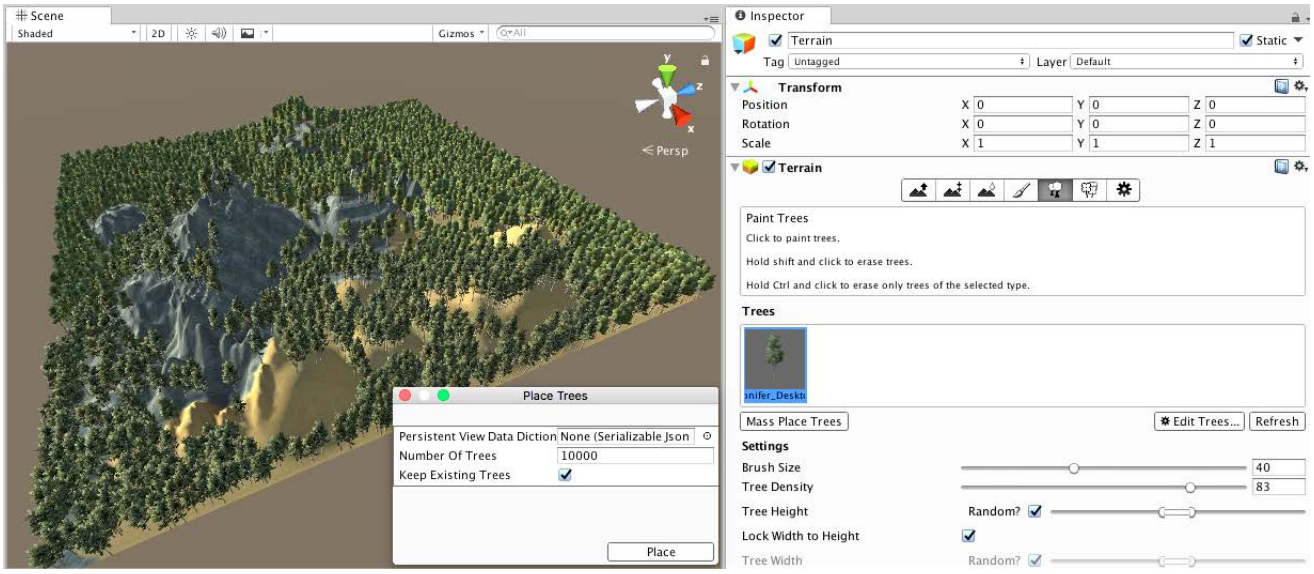


Bild 29: Funktion „Mass Place Trees“ verteilt die Vegetation bereits automatisch

Für weitere Details der Umgebung oder eine akribisch genaue Platzierung der Objekte darin, muss der Designer in der Regel manuell die einzelnen Objekte, hier im Bild Felsen, in die Szene importieren und deren Position, Rotation und Grösse festlegen.

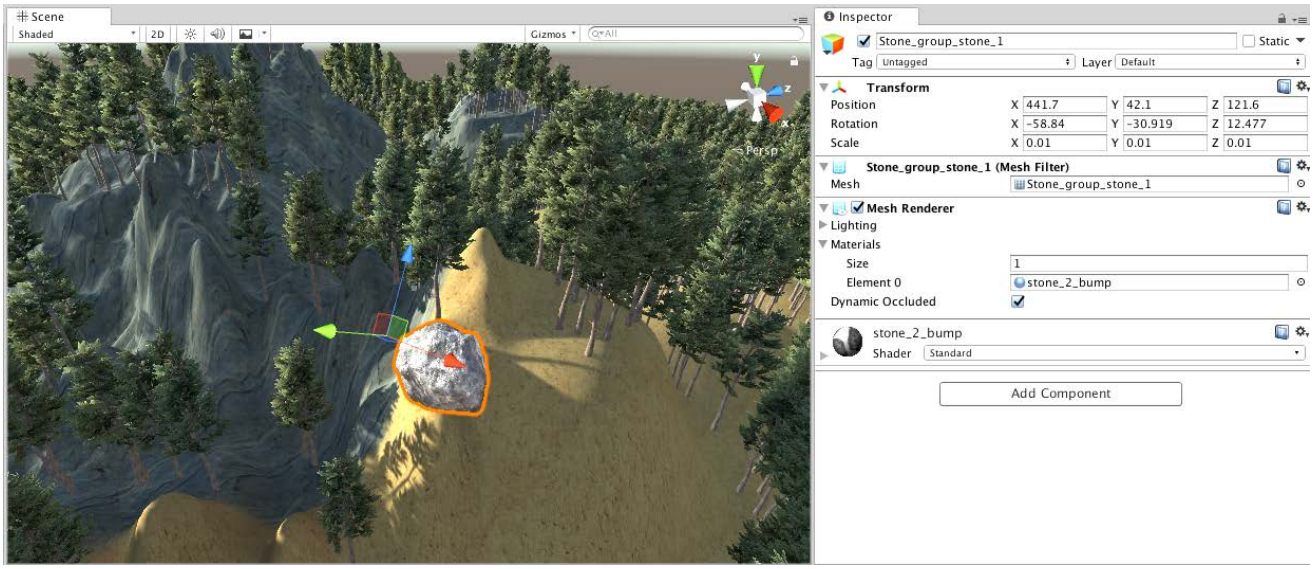


Bild 30: Exaktes Platzieren eines 3D-Felsmodells per Neigungswinkel und Koordinaten

Diese Form der manuellen Umweltgestaltung ermöglicht dem Designer sehr präzise Arbeit im dreidimensionalen Raum, jedoch ist es auch enorm zeitaufwendig und kann bisweilen ohne zusätzliche Erweiterungssoftware (Plug-Ins) nur auf einem Terrainobjekt pro Szene angewendet werden. Für ein Open-World-Game wäre theoretisch ein Terrainobjekt unglaublich grosser Dimension nötig. Dieses Riesen-Terrainobjekt müsste beim Start des Games bereits vollständig geladen werden, was auf heutigen PC-Systemen schlicht nicht realistisch ist.

2.1.2 Manueller Designprozess von Lebewesen (Creature-Design)

Wenn ein Designer eine Kreatur für ein Videospiel zu designen beginnt, stellen sich Fragen zum Story-Kontext, da das Wesen sich nicht nur visuell in das Spiel eingliedern soll. Wo spielt das Game? In der Tiefsee? In einer abstrakten Pixelwelt? Auf den Seiten eines alten Buches? Solche Fragestellungen sind für Designentscheide essentiell, werden hier aber nicht näher untersucht. Wichtig ist hier der Entstehungsprozess des Kreaturen-, und Charakterdesigns.

Der hier beschriebene Prozess wird beim Erstellen der Kreaturen des Prototyps in Kapitel 6 um eigene Techniken erweitert. Als gut dokumentiertes Beispiel zur Verdeutlichung dieser Behauptungen, wird der, zu Demonstrationszwecken erstellte Film „The Blacksmith" in diesem Kapitel genauer beschrieben.[43]

Ein Concept-Artist[44] oder der allein arbeitende Designer beginnt das geplante Konzept erstmal als Silhouette oder Zeichnung umzusetzen und dessen visuelle Wirkung erfahrbar zu machen. Sobald ein Resultat zufriedenstellend ist, wird es als 3D-Modell in einem oder mehreren 3D-Programmen am Computer manuell umgesetzt.

Anschliessend wird das 3D-Modell texturiert und die Gesten der Figur animiert.

Bild 31: Konzept für den Character „Black Smith" von Georgi Simeonov

Bild 32: Umsetzung des Konzepts als 3D-Modell mit Textur von Jonas Thornquist

Dasselbe Prinzip gilt auch für alle anderen Objekte, die in der Spielewelt als dreidimensionale Körper vorkommen (Häuser, Felsen, Bäume), wobei natürlich nicht alle animiert werden müssen.

---

[43] Rasheva, Silvia: Making of The Blacksmith: Concept and Art Production (2015) https://blogs.unity3d.com/2015/06/15/making-of-the-blacksmith-concept-and-art-production/ (30.05.2018)

[43] Concept Artis = Illustrator/in, die/der ein Konzept für Spielinhalte visuell umsetzt, bevor es in die finale Produktion geht. Wikipedia. Begriff: Concept Art https://de.wikipedia.org/wiki/Concept_Art (09.06.2017)

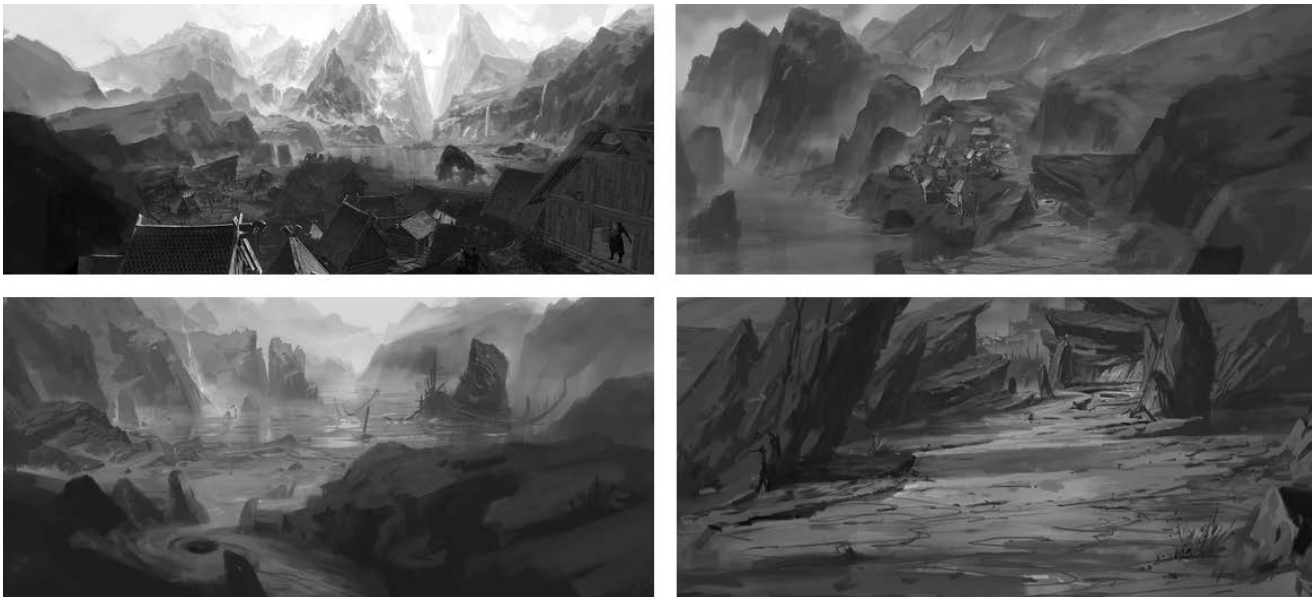


Bild 33: Konzeptskizzen zur Umwelt des Charakters

Um gute und kohärente Resultate zu erhalten ist beim manuellen Designprozess enorm wichtig stets die Bildsprache zu überprüfen und in einer grösseren Produktion mit vielen einzelnen menschlichen Akteuren die Gestaltungsrichtlinien gut und klar zu kommunizieren.

2.2 Generierungsprozess von digitalen Spielwelten vor oder während des Spielens

Im Gegensatz zum manuellen Designprozess, welcher hier ebenfalls als Generierung verstanden wird, hat prozedurale Generierung den Vorteil, quasi kurz vor dem Moment zu wirken, in dem das Designresultat im Spiel wichtig ist (siehe Kapitel 1.7).

Eine Auswahl an markanten Vorteilen beim Laden vor dem effektiven Spielen, im Vergleich zum Generierungsprozess während des Games sind:

- Kein ladebedingtes Einfrieren des Bildschirms (Freeze[45])
- Keine ladebedingten Verzögerungen des Spielflusses (Lags[46])
- In Unity kann die Lichteinstrahlung/Lichtbrechung auf statischen Objekten in der Spielwelt vor dem Export des Games vorberechnet werden (baked Lighting[47]) , was Rechenleistung einspart.

[45] Mit Einfrieren ist das komplette stillstehen des Games auf visueller Ebene gemeint. Während des, auch „Freeze" genannten, Einfrierens reagiert der Computer auch nicht auf Inputs seitens des Spielers, was sehr frustrierend ist und bei längerem Anhalten den Spielfluss völlig zerstört.

[46] Als Lag bezeichnet man die verzögerte Darstellung des Games auf visueller Ebene und auch das verzögerte Reagieren der Spielelemente auf den Steuer-Input des Spielers. Das Spiel friert dabei aber nicht vollständig ein.

[47] Baked Lighting zu nutzen bedeutet, anstelle von Echtzeit-Licht-, und Schattenberechnungen, diese bereits vorberechnet in das Spiel einzugliedern

Eine Auswahl an markanten Nachteilen beim Laden vor dem effektiven Spielen des Games sind:

- Die Ladezeit vor Beginn des Spiels ist durch die Datengrösse der Inhalte definiert und kann sehr lange dauern
- Aufwendig gestaltete, dreidimensionale und zusammenhängende Spielwelt können in ihrer Dimension nicht extrem gross sein, da die Generierung vom Rechner nur sehr langsam oder gar nicht zu bewältigen ist. Es müssen aus Speicherkapazitätsgründen einzelne Level kreiert werden, die anschliessend von Neuem geladen werden und den Spielfluss beeinträchtigen können.
- Alle Inhalte müssen vor dem Spielstart generiert werden, was eine Adaption der Inhalte an spezielle Spielgegebenheiten oder Situationen nur begrenzt ermöglicht.

Ob nun die Generierung vor oder während des Spielens die passendere Lösung darstellt, ist projektabhängig.

2.3 Status Quo: „elektronisch-prozedurales" Spieleweltdesign in Games mit Anwendungsbeispielen

In diesem Kapitel wird eine Auswahl an, von Computern ausgeführten, prozeduralen Generierungsverfahren vorgestellt. Dabei dienen die dazu erwähnten Games als exemplarische Anwendungsfälle.

2.3.1 Generierung von einfachen Welten: Rogue – das erste prozedural erstellte Game

Der schottische Astrophysiker, Informatiker und YouTube-Star Scott Manley schreibt in einem seiner Let's Plays[48] zum 1980 an der kalifornischen Universität von Berkeley entstandenen Spiel *Rogue*:

> „Let's take a look at the origin of the term ‚Roguelike' - the original 1980 text terminal, procedural content, save-free dungeon crawler." [49]

Das Game *Rogue* gilt als eines der ersten Games, welches sich prozedurale Weltengenerierung zu Nutze machte. Wie Bild 34 zu entnehmen ist, sieht diese Welt aus heutiger Sicht grafisch stark reduziert aus. Die Verliesse, die der Spieler (@-Symbol) erforscht (engl. „to crawl" = kriechen, krabbeln) und die im Englischen als Dungeons bezeichnet werden, bestehen, wie die restlichen Elemente auch, aus Textsymbolen. Der Spielstand kann im Originaltitel nicht gespeichert werden, da die Spielwelt jedes Mal neu, prozedural generiert wird. Ein Weg dies heute zu umgehen besteht darin, denselben Seed zu verwenden oder diesen abzuspeichern. Die prozedural generierten Verliesse, Wege, Gegner und Gegenstände (Items) ermöglichten unzählige Varianten von Welten, die auf den damals auch sehr kleinen Speichermedien kaum Platz beanspruchten. Zudem konnten erstmals die Entwickler selbst ihr Spiel spielen ohne die Spielwelt vorher zu kennen, was den Spielspass für sie enorm erhöhte.[50]

---

[48] Let's Play = Vorführen und Kommentieren eines Videospiels. Meist über eine Online-Videoplattform. Wikipedia. Begriff: Let's PLay https://de.wikipedia.org/wiki/Let's_Play (14.05.2018)

[49] Manley, Scott: Rogue – The Original Rogue-like (2013) https://www.youtube.com/watch?v=vxF1osPkplA&t=89s (14.05.2018)

[50] Ebd.

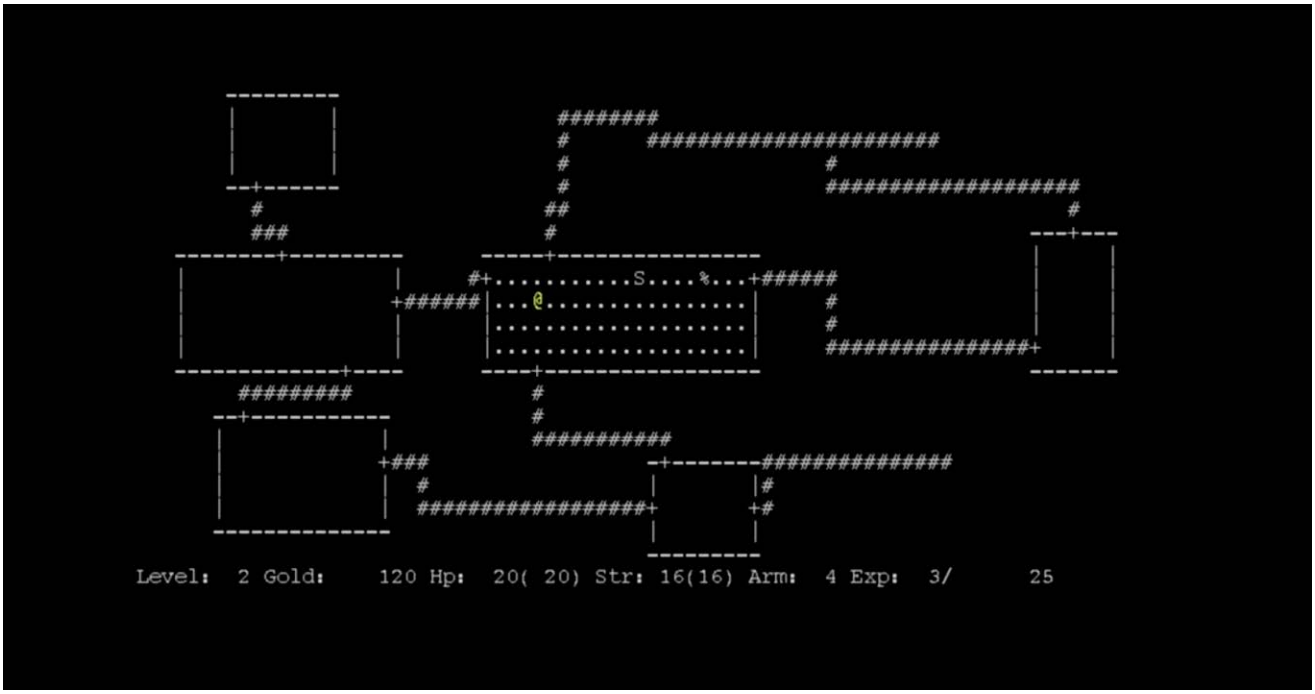


Bild 34: Original Rogue aus dem Jahr 1980

Die Bezeichnung „rogue-like“ wurde genrebezeichnend für alle Spiele, die sich an dieser Verliessstruktur aus Rogue anlehnen und prozedurale Generierung nutzen.[51]

2.3.2 Iterative Generierung von Welten: Dwarf Fortress

Dwarf Fortress [13] gehört zur Familie der rogue-like Games und baut sich iterativ aus AI-agent Proc. Gen. auf. Das 2006, nach vier Jahren Entwicklung, in einer ersten Version veröffentlichte Spiel baut sich also nach und nach auf Basis der vorhergegangenen Resultate des Generierungsprozesses auf. Die Parameter für die prozedural ablaufenden Generierungsprozesse können im Vorfeld vom Spieler definiert werden.

Bild 35: Manipulierbare Werte für die Generierung der prozeduralen Welt von Dwarf Fortress

[51] Wikipedia. Begriff: Rogue-like https://de.wikipedia.org/wiki/Rogue-like (14.05.2018)

> „Satte 20 Minuten werden auch aktuellste Prozessoren gequält, bis das Spiel beim ersten Start eine komplette Fantasywelt generiert hat. Immerhin werden unter anderem Erosion, der Verlauf von Flüssen, geologische Ablagerungen und zudem noch eine tausendjährige Fantasy-„Geschichte“ generiert, um eine stimmige, insgesamt einige 1000 Quadratmeilen umfassende Welt zu erstellen, in der man im „Adventure Mode“ entweder herumwandern oder, sozusagen als Hauptspiel, eine ausgefeilte Aufbaustrategiesimulation spielen kann, als Mentor einer wachsenden Gruppe von nicht direkt steuerbaren Zwergen, die sich als Pioniere auf einem sorgfältig ausgewählten Fleckchen irgendwo in der riesigen Welt ein neues Heim bauen sollen.“ [52]

Man mag bei der Zeitangabe von 20 Minuten staunen, wenn man diesen Erlebnisbericht eines Spielers liesst, jedoch stammt der Text aus dem Jahr 2008 und kann noch heute in Relation mit der Zeit, die bei manuellen Verfahren für ein ähnliches Resultat benötigt werden würde, als extrem effizient betrachtet werden. Auch das Stichwort einer „stimmigen Welt“ aus dem Zitat macht die Wichtigkeit einer kohärenten Welt für den Spieler klar.

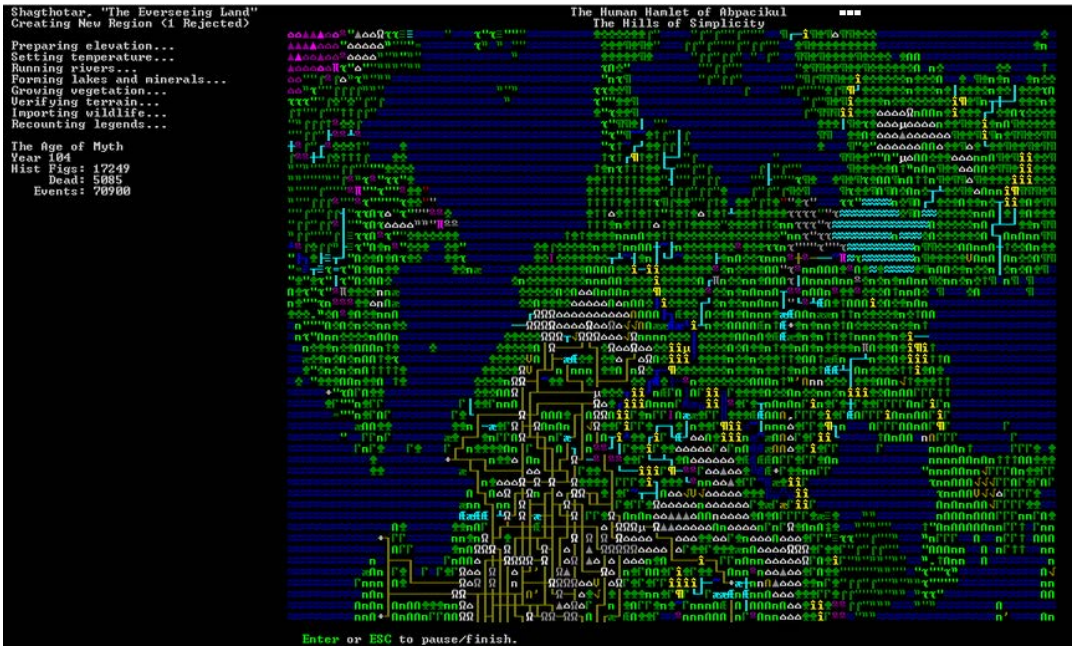


Bild 36: Iterativ generierte Welt aus Symbolen, die jeweils eigene Eigenschaften und Werte darstellen

2.3.3 Vor-, und Nachteile bei prozedural erstellten Gamewelten: No Man’s Sky

In Kapitel 1.7 wurde *No Man’s Sky* bereits vorgestellt. In diesem Kapitel wird besonderes Augenmerk auf die Vor-, und Nachteile von prozeduraler Generierung in Bezug zu *No Man’s Sky* gelegt.

[52] Sigl, Rainer: Spielerlebnis für Hartgesottene (2008) https://www.heise.de/tp/features/Spielerlebnis-fuer-Hartgesottene-3416887.html (14.05.2018)

2.3.3.1 Elite (1984) – Der „No Man's Sky-Vorreiter“

Der Vollständigkeit halber muss in einem Diskurs über *No Man's Sky* auch auf *Elite* aus dem Jahr 1984 verwiesen werden. Das Spiel, eine Wirtschafts-, und Weltraumsimulation, gilt als ein früher Vertreter des Open-World-Genres.[53]

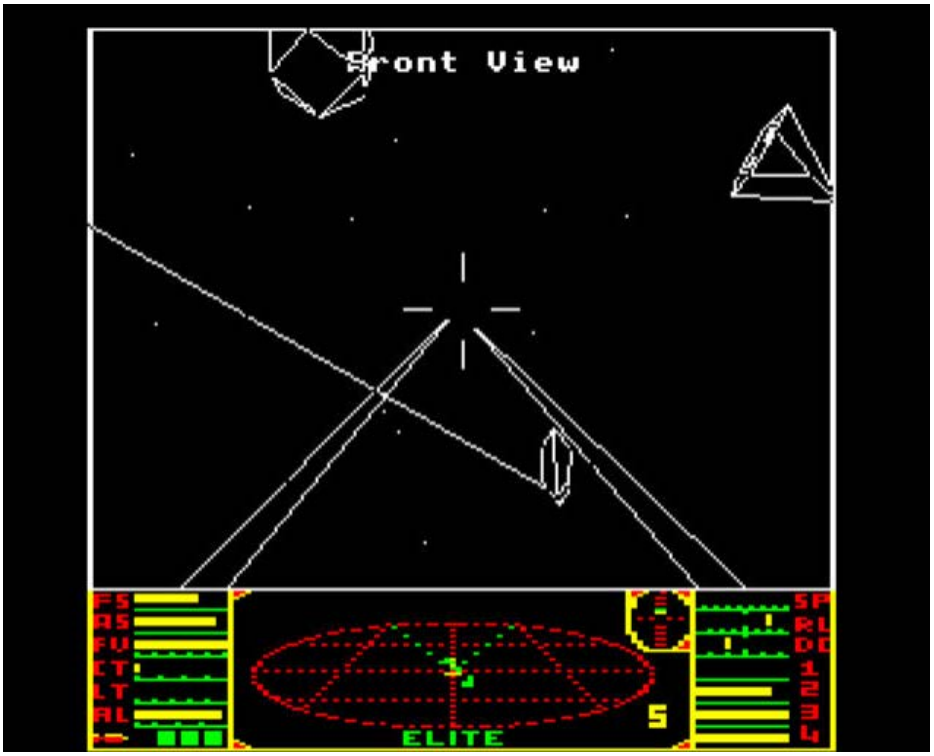


Bild 37: Blick aus dem Raumschiff-Cockpit in ein generiertes Universum in Elite

2.3.3.2 Terraingenerierung in No Man's Sky

Eine detaillierte Auseinandersetzung mit der prozeduralen Generierungstechnik aus dem Weltraum-Erkundungsgame *No Man's Sky* ist weit mehr, als im Rahmen dieser Arbeit möglich ist. Ich verweise deshalb auf ein einstündiges Referat von Innes McKendrick, einem Entwickler von *No Man's Sky*, der an der jährlichen Spieleentwicklerkonferenz GDC[54] in San Francisco über genau diese Thematik spricht. Folgend eine Zusammenfassung besagten Referates.[55]

No Man's Sky nutzt sowohl prozedurale Generierung vor, als auch während der Spieler sich in der Spielwelt, also den Planeten in Kugelform, frei bewegen kann. Zu Zweiterem gehört das Generieren des Terrains, auf dem der Spieler sich gerade in Sichtweite befindet. Das Terrain wird auf Basis einer sogenannten Noisemap erstellt. Noisemaps sind meist prozedural erstellte Bilddateien, welche in der Computergrafik in Form von Heightmaps (Höhenfelder) wie Landkarten mit Höhenkurven, dazu genutzt werden, Höhenunterschiede einer 3D-Struktur zu erstellen. In Kapitel 1.5 wurde dies bereits bebildert und beschrieben. Auf Basis des so entstandenen Terrains wird eine Voxelwelt aufgebaut. Voxel sind einzelne 3D-Würfelstrukturen, die diverse Daten, wie Punkte in einem Koordinatensystem, aber auch Farbinformationen, etc. speichern können.

---

[54] Offizielle Website der GDC http://www.gdconf.com (14.05.2018)

[55] McKendrick, Innes: Continuous World Generation in No Man's Sky. Digitally Speaking (2017) https://www.gdcvault.com/play/1024265/Continuous-World-Generation-in-No (14.08.2018) 8min

Bild 38: Innes McKendrick an der San Francisco Game Developers Conference (GDC) über die Voxel in No Man's Sky

Daraus ergibt sich für das Terrain in *No Man's Sky* folgende Generierungshierarchie:

1. Daten als Noisemap generieren
2. 3D-Landschaftsstruktur aus gegebenen Noisemap-Daten erstellen
3. 3D-Landschaft zu Kugelform (Planet) umrechnen
4. Kollisionskörper (Collider[56]) zur Landschaft hinzufügen
5. Künstliche Intelligenz für Findung von begehbaren Regionen für Kreaturen nutzen
6. Landschaft mit Vegetation und Kreaturen besiedeln

2.3.3.3 Nutzung der "Superformula" in No Man's Sky's Terraingenerierung

Sean Murray, Mitbegründer des Studios Hello Games, welches *No Man's Sky* entwickelte, stiess laut einem in The New Yorker veröffentlichtem Artikel bei der Generierung von verschiedenen Terraintypen bald auf ein Problem:

> „One of the hardest things for us to do is to create coherent shapes." [57]

Die Lösung bat ihm eine geometrische Figur namens „Superformula", welche vom belgischen Biologen Johan Gielis entwickelt wurde. Diese Kurvenformel ist in der Lage sehr einfach symmetrische und vegetative Formen zu berechnen. Um das Potential dieser generativen Formensprache zu verdeutlichen, wird sie hier mit Illustrationen des Zoologen Ernst Haeckel gezeigt.

---

[56] 3D-Netzstrukturen wie die 3D-Landschaft werden als Mesh bezeichnet. Meshes existieren rein visuell, d.h. sie haben selbst keine einprogrammierte Registrierung, ob sie von einem anderen Objekt berührt werden. Dazu müssen dem Mesh sogenannte Collider, im Idealfall mit dem Mesh deckungsgleiche 3D-Strukturen ohne visuelle Repräsentation, eingefügt werden. Collider registrieren Kontakte mit anderen Collidern im Game.

[57] Sean Murray zitiert von Khatchadourian, Raffi: World Without End. Creating a full-scale digital cosmos. The New Yorker (2015) https://www.newyorker.com/magazine/2015/05/18/world-without-end-raffi-khatchadourian (14.05.2018)

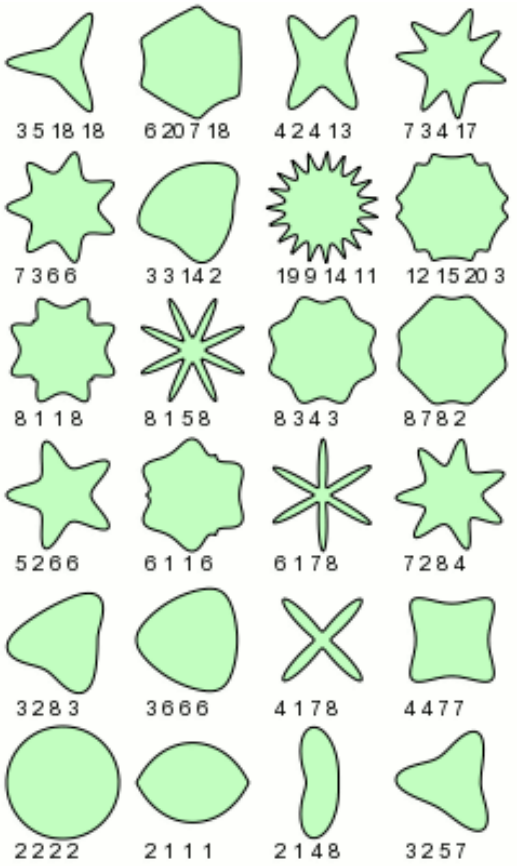


Bild 39: Mit der Superformula generierte Formen

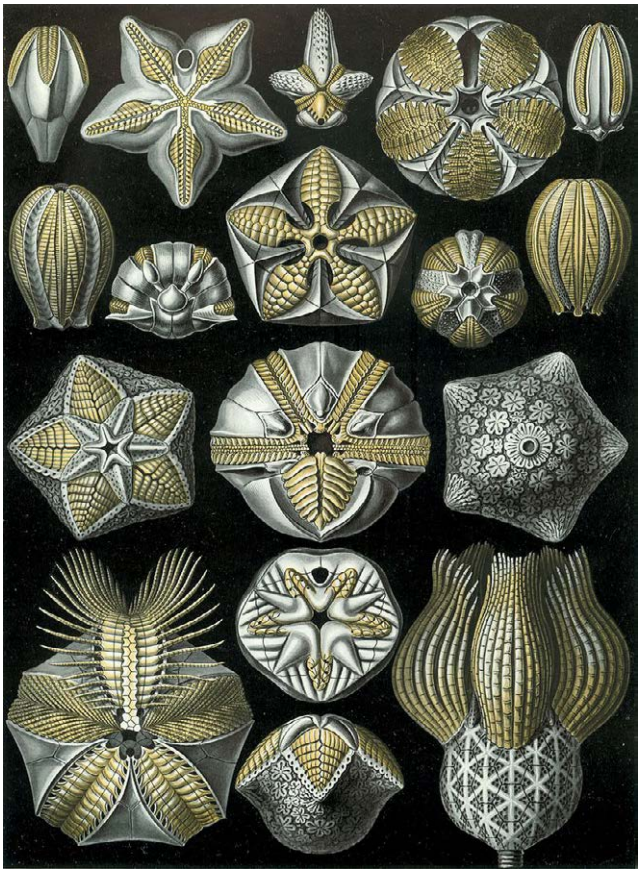


Bild 40: Illustration von Blastoidea (Knospenstrahlern) aus Haeckel's Kunstformen der Natur (1899)

> „He (Sean Murray) envisioned using the Superformula throughout the game, especially at the center of the galaxy, where landscapes would become more surreal. With only small shifts in its parameters, the equation was producing impressive variability. In one rendering, it produced rolling hills. Murray refreshed the screen: a star-shaped rock formation appeared. He seemed pleased." [58]

2.3.3.4 Vegetationsplatzierung in No Man's Sky

Zur Vegetationsgenerierung sagt McKendrick sinngemäss wiedergegeben gegen Ende des Talks[59], dass die Vegetationsplatzierung von künstlerischen Kriterien getragen wird. Er beschreibt, im Kontext des Umwelt-Designs von *No Man's Sky*, wie ein alleinstehender Baum in einer Landschaft eine weniger starke ästhetische Wirkung zeigt, als ein Baum zusammen mit einer Gruppe aus kleiner werdenden Vegetationseinheiten. Es gilt die Annahme, dass diese Entscheide auf der natürlichen, visuellen Anordnung von Vegetation auf der Erde beruht. Dies wird unter Zuhilfenahme einer schematischen Unterteilung des Terrains in ein Gitter klar, welches die Position der grossen und kleinen Vegetationseinheiten definiert.

---

[58] Khatchadourian, Raffi: World Without End. Creating a full-scale digital cosmos. The New Yorker (2015) https://www.newyorker.com/magazine/2015/05/18/world-without-end-raffi-khatchadourian (14.05.2018)

[59] McKendrick, Innes: Continuous World Generation in No Man's Sky. Digitally Speaking (2017) https://www.gdcvault.com/play/1024265/Continuous-World-Generation-in-No (14.08.2018) 47:00 min

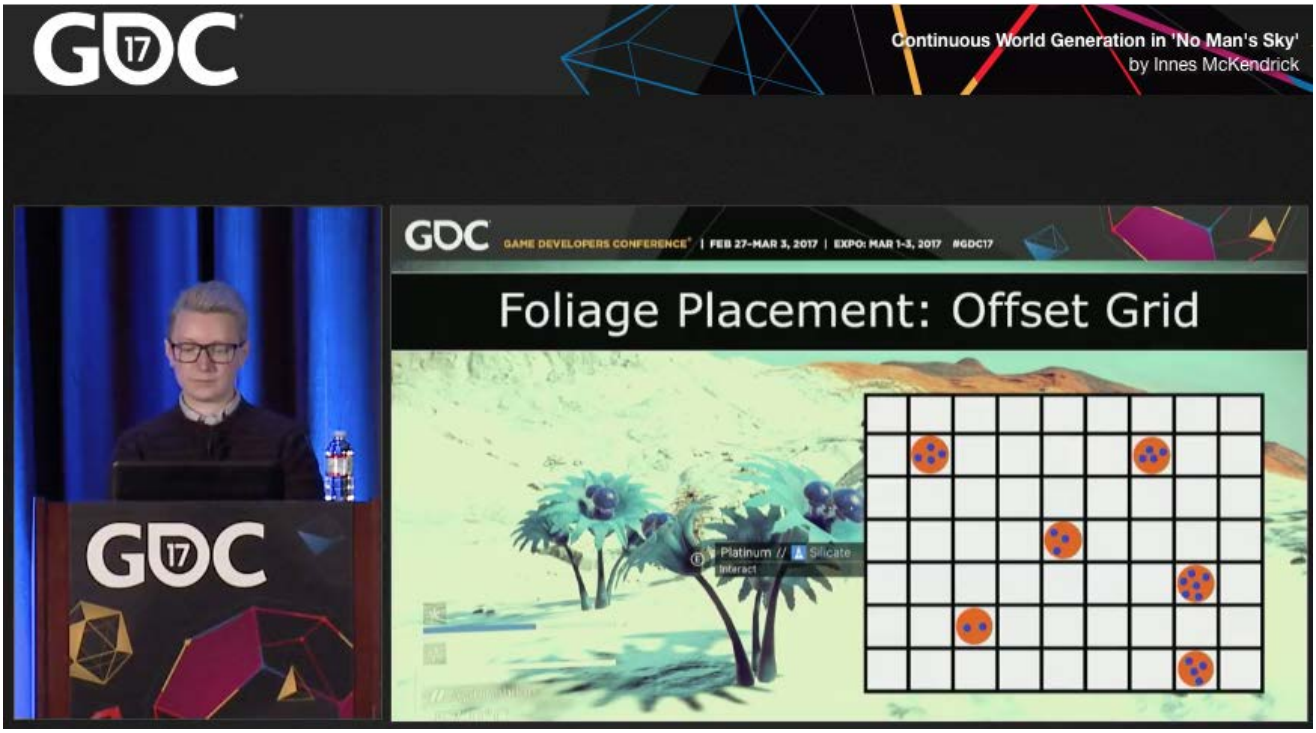


Bild 41: Innes McKendrick an der GDC über Vegetationsanordnungen in der Welt von No Man's Sky

2.3.3.5 Wesen-, bzw. Aliengenerierung in No Man's Sky

No Man's Sky nutzt selbst für das Design von den darin vorkommenden Alienwesen prozedurale Generierung. Auf der Website gamerant.com (Stand 2018) wird unter anderem folgend über diesen Prozess berichtet:

> „Hello Games created a small set of assets, which were used to create hundreds of animals, all with different variants and styles. This was done through procedural generation. Each creature starts as a base model blob with each region of its body classified a certain way. From there, the game's code randomly selects an asset that matches that classification, and sticks it in the necessary spot. The final result is hundreds and thousands of different creatures, each with its own design and flare." [60]

Des Weiteren hat Sean Murray, Kopf des Projektes *No Man's Sky*, laut der Website inverse.com, bei einem Interview am Sitz der Europäischen Weltraumorganisation ESA, zusätzliche Informationen freigegeben:

> „We've invented a system that automatically balances out the weight and adjusts the skeleton." [61]

[60] Blake, Boston: No Man's Sky Planet and Creature Creation Shown Off (2016) https://gamerant.com/no-mans-sky-planet-creature-creation/ (14.05.2018)

[61] Murray Sean zitiert von Kratsch Benjamin: Sean Murray Says Yes to Space Whales (2016) https://www.inverse.com/article/19373-sean-murray-interview-no-man-s-sky-space-whales-elon-musk (14.05.2018)

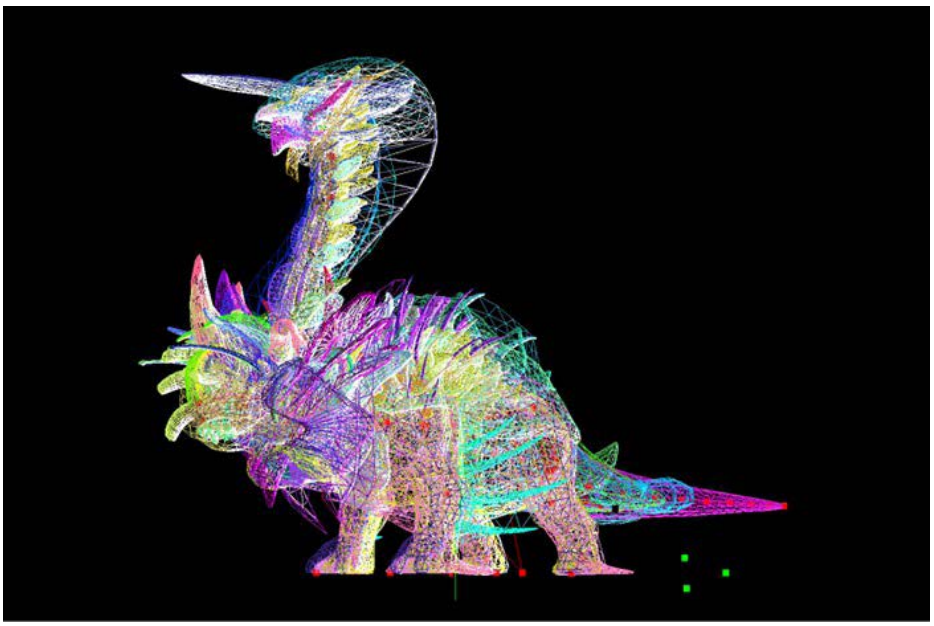


Bild 42: Sich Überlagernde 3D-Strukturen zur Kombination für Kreaturen

Es ist deutlich spürbar, dass prozedurale Techniken und Automatisierung im Designprozess der Kreaturen bei *No Man's Sky* eine grosse Rolle spielen. Und obwohl McKendrick an seinem Referat betont, dass es nicht darum geht die Designer im Entwicklungsteam zu ersetzen, sondern ihnen zu helfen, schneller mehr Inhalte zu produzieren[63], so sind doch immer wieder ähnliche Vorwürfe eines nicht glaubwürdigen Designs auf Grund fehlender menschlicher Evaluation (wie sie bei human-agent Proc.Gen der Fall wäre) bezüglich ihrer Ästhetik zu lesen.

> „Granted, while this method has the potential to create beautiful creatures of wonder, it can also spit out a hilarious, circus-worthy animal that seems to defy gravity thanks to its funky proportions." [64]

Bild 43: Bild 42: Eine etwas quirlig wirkende, saurierähnliche Kreatur aus No Man's Sky, die in dieser Form wohl kaum ein Designer im Kontext eines nicht-humoristischen Spiels veröffentlichen würde

[63] McKendrick, Innes: Continuous World Generation in No Man's Sky. Digitally Speaking (2017) https://www.gdcvault.com/play/1024265/Continuous-World-Generation-in-No (14.08.2018) 54:50 min

[64] Blake, Boston: No Man's Sky Planet and Creature Creation Shown Off (2016) https://gamerant.com/no-mans-sky-planet-creature-creation/ (14.05.2018)

3. Auswahl kohärenter Welten in Literatur und Film

In folgendem Kapitel wird sehr generell auf die Auseinandersetzung mit Literatur-, und Filmbeispielen eingegangen, die für die Entwicklung des Prototypens dieses Projektes als Inspirationsquelle dienten. Für eine Ausführung sind jeweils die Verweise zu den Originalquellen zu beachten.

3.1 Verhältnis von Mensch zu Umwelt – anhand von Topophilia (Yi-Fu Tuan)

Der Band Topophilia – A Study of Environmental Perception, Attitudes and Values[64] des Autors und Humangeografen Yi-Fu Tuan ist hinsichtlich des Verhältnisses eines Menschen gegenüber der ihn umgebenden Umwelt und im Besonderen der Landschaftsform eine erwähnenswerte Ressource. Diesen Diskurs führt der Autor aus egozentrischer (individueller) und ethnozentrischer (einer ethnischen Gruppe zugehöriger) Perspektive. Der Term „Topophilie" beschreibt eine starke, emotionale Zuneigung eines Menschen zu einer Landschaft. Zusammenfassen ziehe ich für diese Arbeit folgende, meiner Ansicht nach relevante Schlüsse:

- Es gibt keine Landschaftsform, die bei jedem Menschen Topohilie auslöst
- Die Wahrnehmung, Wertschätzung und Haltung gegenüber der Umwelt ist bestimmt durch die (kulturelle/kulturgeschichtliche) Erziehung. Beispiel: Begriff von „Wildnis" ist zur Zeit der Industrialisierung romantisiert, zuvor, zur Zeit der Besiedlung des Amerikanischen Mittellandes durch europäische Siedler, ist „Wildnis" eine primär gefährliche und furchteinflössende Umgebung.
- Die Wahrnehmung, Wertschätzung und Haltung kann sich im Laufe des Lebens eines Menschen verändern
- Wald, Küste, Tal und Insel sind Begriffe, die allgemein positive Assoziationen auslösen und in Paradiesvorstellungen weltweit eine grosse Rolle spielen

In Games können diese Annahmen, die Gemeinsamkeiten mit den formalen Gestaltungsprinzipien aus Kapitel 1.6.1 aufweisen, womöglich helfen dem Spieler gewisse Stimmungen zu vermitteln.

3.2 Verhältnis von Menschen zu Elementen der Umwelt – anhand von „Poetik des Raumes" (Gaston Bachelard)

Das Werk des französischen Philosophen, Dichters und Wissenschaftstheoretikers Gaston Bachelard beinhaltet Textstellen, welche im Kontext von Games durchaus bedeutsam sind, da sie auf die Wirkung von Umwelt und deren Elementen auf den Menschen eingehen. Seine Aussagen über Welt werden für die Arbeit auf den Kontext von Spielewelten bezogen. So schreibt Bachelard:

„Indem man kleine Probleme löst, lernt man grossen beizukommen." [65]

[64] Tuan, Yi-Fu: Topophilia – A Study of Environmental Perception, Attitudes and Values. Englewood Cliffs. New Jersey. Prentice Hall Inc. (1972)

[65] Bachelard, Gaston: Poetik des Raumes. Frankfurt am Main. (1957) S.143

Dieses Zitat kann genauso für den Einstieg in ein Videospielen übernommen werden, da auch dort der Spieler im Regelfall nach dem „Versuch und Irrtum“ (Trial and Error)-Prinzip die Welt erfährt und sich zum Ziel vorarbeitet. Die Rolle der kleinen Details der (Game)Welt wird somit hervorgehoben. Bachelard schreibt ausserdem über die Wichtigkeit des „Geheimnisses“ in einer Welt.[66] Dies trifft auch in Open-World-Games zu, die Spieler motivieren, die Spielwelt zu erkunden.

> „Für Bachelard bedeutet dies den Raum, als erfahrbares Etwas, in all seinen empfindungsmäßigen Konsequenzen zu untersuchen. Somit sind die, von der Einbildungskraft, erfassten Räume keine indifferenten, für sich alleinstehenden Räume die nur den geometrischen Maßeinheiten zuzurechnen sind, sondern erlebbare Ausgedehntheiten, die nicht nur in ihrer Positivität erlebt werden.“ [67]

Der erlebbare Raum ist ein interessanter Aspekt, der sich auf den interaktiven Raum in Games übertragen lässt.

3.3 Der interaktive Raum – anhand von „Stalker“ (Andrei Tarkowski)

Der Film *Stalker*[68] gilt als Klassiker des sowjetischen Kinos und war wegweisend für das Genre des Science-Fiction. Im Film begeben sich mehrere männliche Protagonisten in eine verbotene und tödliche „Zone“, die einen wunscherfüllenden Raum beinhaltet. Während die Männer den Raum erkunden, verändert dieser sich für sie erfahrbar.

Bild 44: Die drei Charaktere in einem der geheimnisvollen Räume in „der Zone“

---

[66] Bachelard, Gaston: Poetik des Raumes. Frankfurt am Main. (1957) S.97

[67] Bachelard, Gaston: Poetik des Raumes (1957). In: Dürre, Jörg/ Günzel, Stephan (Hrsg.): Raumtheorie. Grundlagen aus Philosophie und Kulturwissen-schaften. Frankfurt am Main. Suhrkamp (2006) S.167

[68] Tarkovski, Andrei: Stalker. Mosfilm (1978/79)

Die Übersetzung des interaktiven Raumes von *Stalker* in ein Spielprinzip könnte das rätselorientierte Spiel *Antichamber* [14] von Alexander Bruce sein, da die Räume darin eine Vielfalt an Rätselmöglichkeiten zu Tage fördern, die die Umgebung des Spielers als Ausgangslage nutzen.

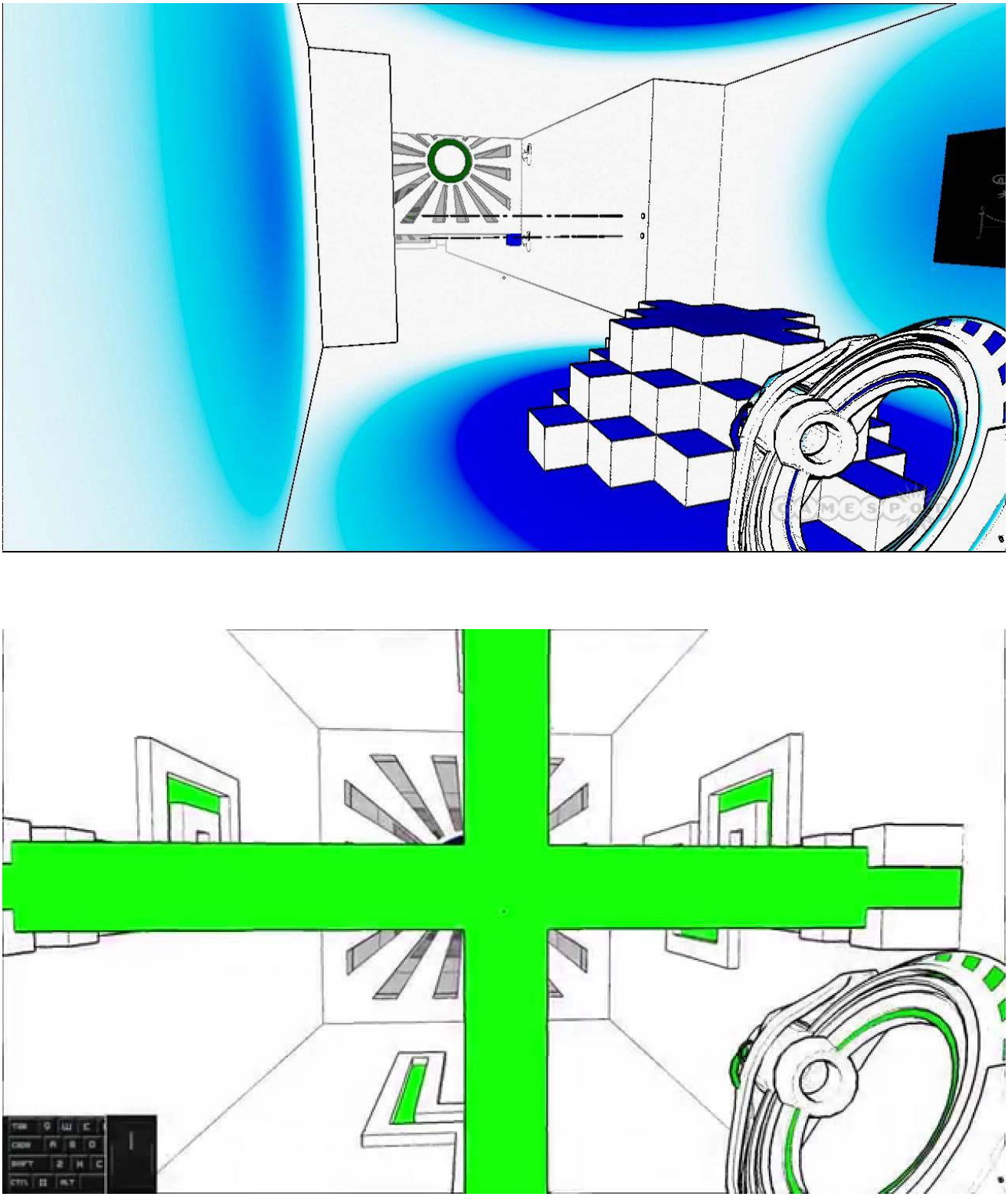


Bild 45: Im Spiel Antichamber werden der 3D-Raum und optische Täuschungen, die nur auf dem 2D-Bildschirm wahrnehmbar sind, zur Basis für diverse Rätsel

Das 2017 erschienene Videospiel *Hellblade: Senua's Sacrifice* [16] erzählt die Heldenreise einer Kriegerin der Pikten (schottisches Volk um das 5. Jh.), die an einer Psychose leidet.[69] Die Fantasy-Momente im Game spielen sich dabei nur im Kopf der Protagonistin ab, was besonders im Audiodesign des Games (Stimmen in Kopf der Protagonistin) umgesetzt ist. Bei *Stalker* adaptiert die Umwelt der Menschen innerhalb der Zone ebenfalls das Innenleben der Figuren und lockt sie so in gefährliche Situationen.

Bild 46: Die Heldin Senua hört Stimmen, die sie ermutigen ihre Reise im Spiel fortzusetzten, sie warnen oder beleidigen

4. Erkenntnis

Hier nun die relevantesten Erkenntnisse aus der bisherigen Analyse für die weiterführende Entwicklung des Game Prototyps *GenoTerra*.

4.1 Fazit der Bestandsanalyse

Prozedurale Generierung gewinnt zunehmend an Bedeutung und kann, wie *No Man's Sky* als aktuellstes Beispiel in Kapitel 2.3.3 (und folgende) verdeutlicht, inzwischen für die Generierung ganzer Spielwelten genutzt werden. Open-World-Games bilden dabei ein Genre, welches besonders von diesen Methoden profitieren kann. Ausserdem gibt es diverse Einsatzgebiete und auch genauso breit angelegte Verfahren, wie prozedurale Generierung eingesetzt werden kann. Eine enge Auswahl von, für diese Arbeit nützlichen prozeduralen Generierungsmethoden, wurde in Kapitel 1.4.1 vorgestellt. Essentiell ist die Feststellung, dass das Thema der visuellen Kohärenz der Spielewelten die Designer immer noch vor grosse technische und ästhetische Herausforderungen stellt.

> Es ist wünschenswert, eine Methode zu entwickeln, bei der die Vorteile von prozeduraler Generierung die Designer so unterstützen, dass diese schneller und einfacher kohärente Spiel-, bzw. Bildwelten in Open-World-Games entwickeln können.

[69] Kamen, Matt: How ‚Hellblade' explores real-life mental health issues (2015) http://www.wired.co.uk/article/hellblade-ninja-theory-mental-health (15.05.2018)

4.2 Eine Formel für alle Fälle – Die Shape Based TFFT-Methode (Terrain, Flora, Fauna, Texture)

Um bei besagter Problemstellung aus Kapitel 4.1 einen strukturellen Lösungsablauf zu ermöglichen, der immer gleich angewendet werden kann und sich somit auf die Kohärenzkriterien aus Kapitel 1.6.2 bezieht, wurde im Rahmen dieser Arbeit die ***Shape Based TFFT-Methode*** entwickelt. Die Methode beschreibt folgenden Ablauf:

1. **T**erraingenerierung anhand von human-agent Proc.Gen-Shapes
2. **F**loragenerierung anhand von human-agent Proc.Gen-Shapes
3. **F**aunagenerierung anhand von human-agent Proc.Gen-Shapes
4. **T**exturgenerierung anhand von human-agent Proc.Gen-Shapes für die Resultate der vorangegangenen Generierungsprozesse

Die Abläufe der human-agent Proc.Gen decken-, oder ähneln sich vom Ablauf her stark. D.h. sie führen auch visuell zu sehr ähnlichen Resultaten. Mehr dazu unter Kapitel 6.

Die ***Shape Based TFFT-Methode*** ist der Versuch, im Kontext der Open-World-Games, ein Hilfsmittel zur kohärenten Spiel-, bzw. Bildsprachengenerierung im Designprozess zu sein.

5. Das Shape-Generator-Tool – Eine Lösung für Kohärenz im frühen Designprozess von Games?

Dieses Kapitel befasst sich mit der Entwicklung einer selbst erarbeiteten, formengenerierenden Software in Unity. Shape-Generator genannt. Wenn von Zeichnen oder Zeichnung die Rede ist, steht der Begriff im Zusammenhang mit digitalem Zeichnen auf einem Grafiktablett.

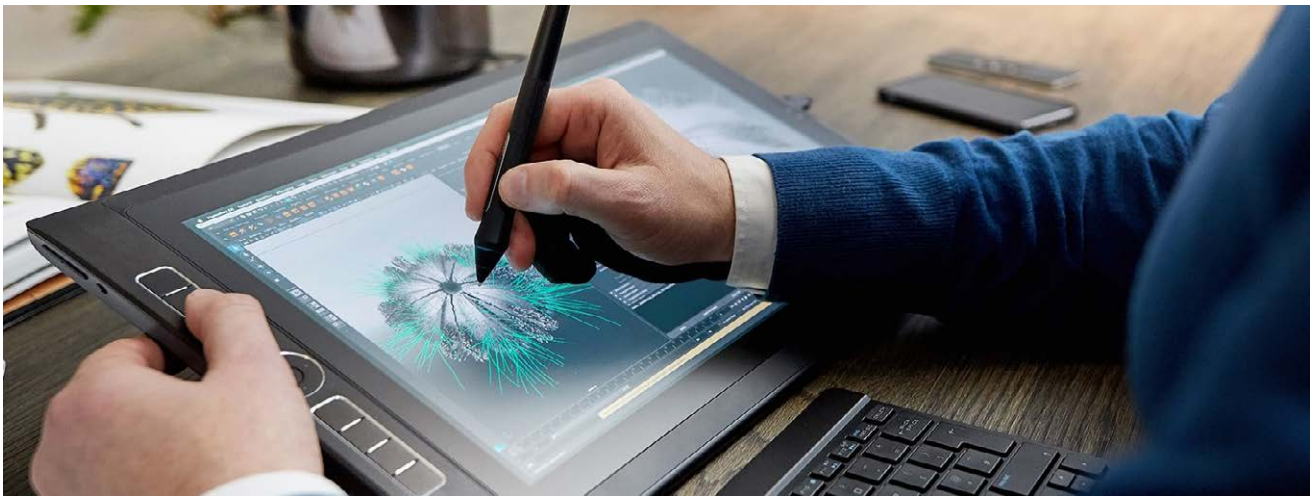


Bild 47: Ein Grafiktablett zum digitalen Illustrieren, Malen, 3D-Modellieren, etc

5.1 Assoziatives Zeichnen als bildsprachengenerierende Technik in der Concept Art

Wie im Development Blog zum Projekt[70] im Detail zu entnehmen ist, wurde im manuellen Designprozess schnell klar, dass eine völlig freie Herangehensweise an das Kreaturendesign meist zu persönlich als unbefriedigend empfundenen Resultaten führte. Die dabei entstandenen Konzeptdesigns für verschiedene Figuren waren zu stark an unbewusste und bewusste Assoziationen gebunden, was zu wenig Variationen führte. „Einmal Gesehenes" überlagert offenbar die Entscheidungen für eine Form. Zu erwähnen ist, das solch praktisch erarbeitete Feststellungen in diesem Kapitel aus der eigenen Erfahrung heraus begründet werden und ihre Allgemeingültigkeit durchaus in einer weiterführenden Arbeit untersucht werden könnte.

Bild 48: Frei gestaltete Kreaturen. Die Designs sind dabei kaum variabel. Es findet ausserdem im Vergleich zu kommenden Beispielen sehr wenig kreative Leistung statt

Eine vermeintliche Lösung fand sich spontan im simplen Kritzeln mit geschlossenen Augen. Dies „generierte" Formen, aus denen Silhouetten herausinterpretiert und mit geöffneten Augen weiter ausgearbeitet werden konnten.

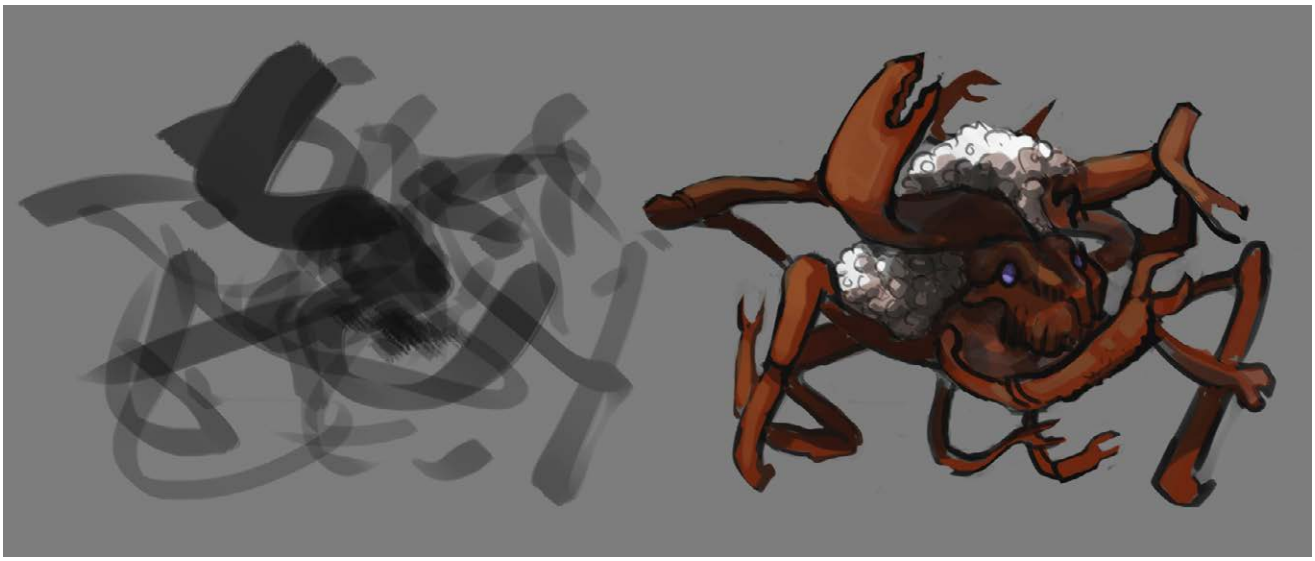


Bild 49: „Gekritzel" (Links) als Ausgangslage für Kreaturen-Konzeptdesign (Rechts)

[70] Siehe Dokumentation Kapitel 10.1 oder www.mnenad.com/creature-design

Bild 50: Bei der Technik des „Assoziativen Zeichnens“ sind auch Duktus und Strichstärke vom Designenden abhängig, was bei einer spielerisch-, freien Herangehensweise ohne strikte Methodik zu weniger einheitlichen Ausgangslagen führt

Nebst der rein visuellen Darstellung wurde auch das Entwickeln kreativer Ideen für Verhaltensweisen von Kreaturen auf dieselbe Art möglich.

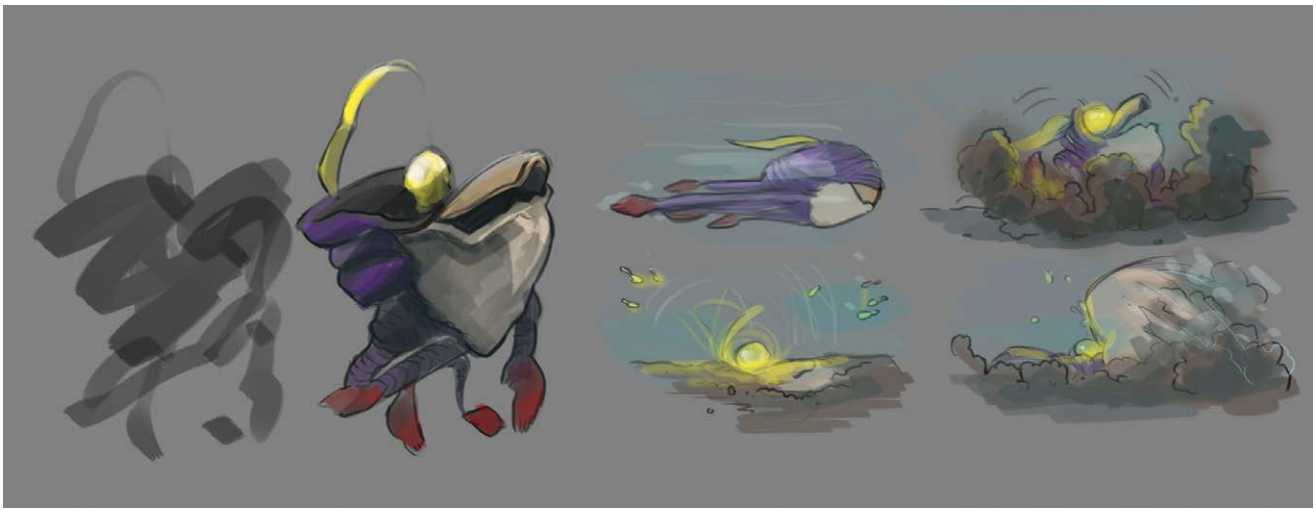


Bild 51: Konzept für eine Tiefseekreatur und deren Lauerjagt (Kreatur vergräbt sich im Sand und lockt durch Licht Beutetiere an, welche sie mit ihrem sackartigen Maul fängt)

Doch auch bei dieser Methode stellte sich schnell heraus, dass sich die Basis der verschiedenen Kreaturen zu stark unterschied, als dass sie wirklich eine kohärente Ausgangslage für eine gemeinsame Spielwelt bilden würde. Im Gegensatz dazu stellte sich sogenanntes „Morphen" als Möglichkeit zur Gewinnung verschiedener Designvarianten aus einem einzigen Ausgangsdesign heraus.

Bild 52: Erstellung von Varianten durch die Verzerrung einer gemeinsamen Ausgangslage

Eine Begründung für die Nutzung dieser Technik des assoziativen Zeichnens fand sich beim US-amerikanischen Gamedesigner Raph Koster. Er setzt das Erkennen von Mustern in Verbindung zu unserem menschlichen Belohnungssystem.

> „Es ist ein zutiefst menschlicher Trieb, Muster zu suchen, zu erkennen und zu lösen. Das Lösen dieser Rätsel oder Erkennen von Mustern wird durch die Ausschüttung von Hormonen wie Endorphinen oder Dopamin belohnt." [71]

Beide Ansätze zur kohärenten Kreaturen-Gestaltung wurden jedoch zu Gunsten einer Generierung mit bereits variablen, jedoch visuell kohärenten Ausgangslagen aufgegeben (siehe Kapitel 5.2). Zu den genannten Annahmen in diesem Kapitel sind im Development Blog ausserdem weiterführende Gedanken formuliert.

[71] Rehfeld, Gunter, Schmidt, Ulrich (Hrsg.): Game Design und Produktion. München. Hanser Verlag (2014) S.18

5.2 Verbesserung des assoziativen Zeichenprozesses durch prozedurale Generierung

Um eine einheitlichere Ausgangslage für den Designprozess zu schaffen, stellte sich prozedurale Generierung als praktikable und effiziente Lösung heraus.

Auf Basis des Codes der prozeduralen Terraingenerierung und Platzierung der Vegetation auf dem Terrain (siehe Kapitel 6.2.1) entstand ein Formengenerator. Dessen Generierungsprozess läuft im Detail folgendermassen ab:

1. Ein Seed generiert mit no-agent Proc.Gen eine Noisemap mit einer klaren Schwarz-Weiss-Trennung (Keine Abstufung)

2. Es werden in der Noisemap Positionen bestimmt, an welchen weitere weisse Formen durch no-agent Proc.Gen generiert werden

3. Weisse Felsen werden erzeugt, die, mithilfe des in Unity bereits gegebenen Physik-Systems[72], auf die Noisemap fallen und sich darauf bis zum Stillstand verteilen

Bild 53: Links: Schwarzweiss Heightmap | Mitte: Weisse Form subtrahiert visuell Teil der Heightmap | Rechts: Weisse 3D-Formen (Rot) werden mit Physiksystem platziert und subtrahieren zusätzlich visuell Teile der Form

Bild 54: Fertige Form (Shape)

[72] Das Physik-System (Physics System) simuliert Gravitation, sowie andere Kräfte in der Software

6. Experiment: „GenoTerra“- ein Open-World-Game mit prozeduralem Inhalt und kohärenten Spielbestandteilen

Dieses Kapitel widmet sich der Entwicklung und Dokumentation eines *GenoTerra* genannten Open-World-Spieleprototypens. Dieses Game ist in Unity entwickelt worden, und beruht zu Teilen auf Open-Source Code von Sebastian Lague.[73] Die Programmiersprache des Projektes ist C# (C Sharp). Die in Kapitel 4.2 vorgestellte ***Shape Based TFFT-Methode*** kommt folgend ebenfalls zur konkreten Anwendung.

6.1 Prozedurale Terraingenerierung

Die prozedurale Generierung der Landschaft basiert auf Sebastian Lagues Quellcode und ist so erstellt, dass Teile der Spielewelt erst dann generiert werden, wenn der Spieler seinen Avatar nahe genug an die Kante des bereits generierten Spielweltstückes (Chunk) lenkt. im Idealfall liegt die Sichtweite im Game ein wenig kürzer als dies Generierungsdistanz, was die Illusion von einer endlosen Landschaft kreiert.

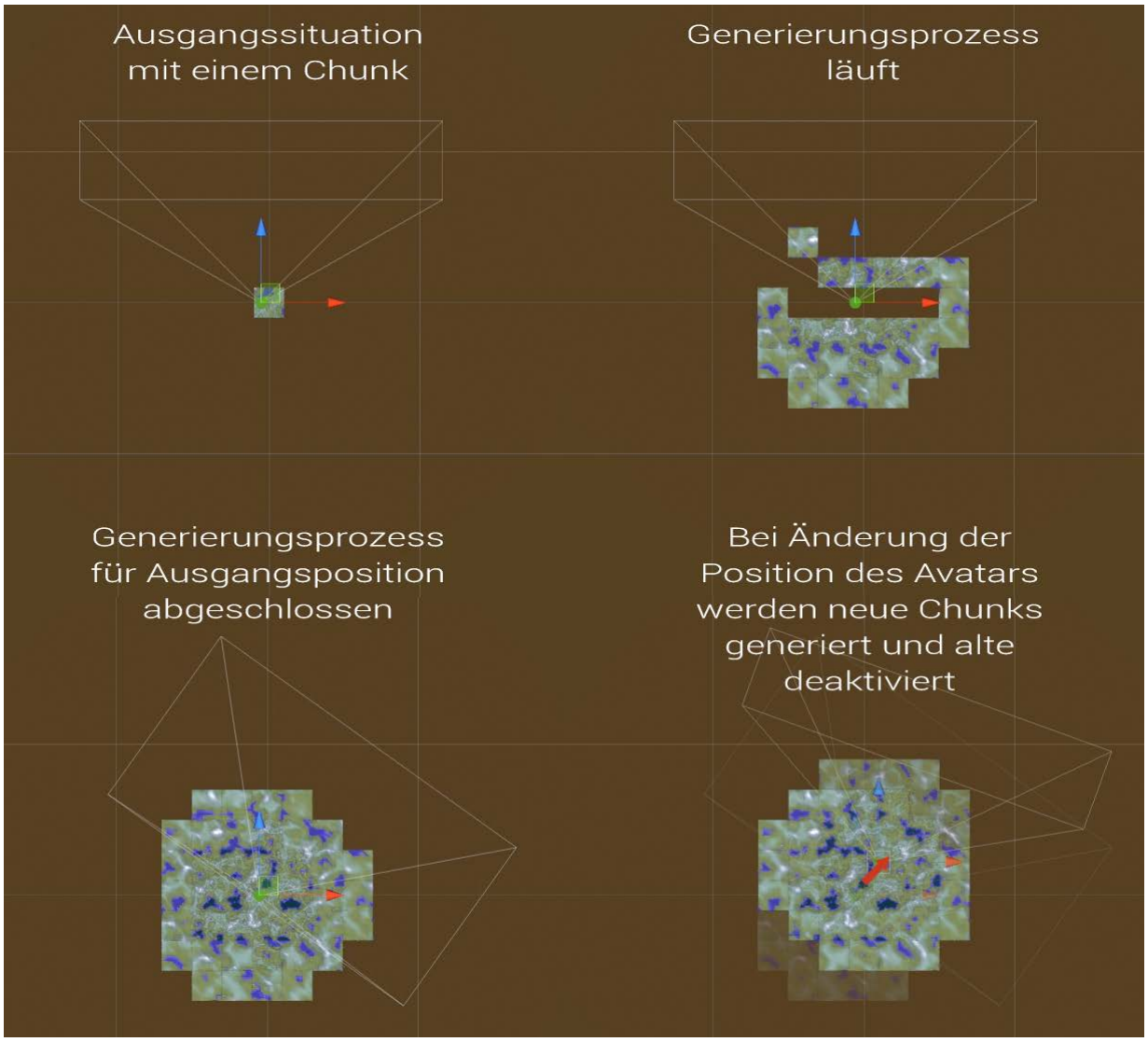


Bild 55: Verdeutlichung der Generierung der einzelnen Terrain-Chunks

[73] Lague, Sebastian: Procedural Landmass Generator (2017) https://github.com/SebLague (15.05.2018)

Die gezeigten Bilder sind im Original animiert und können zur besseren Verständlichkeit zusätzlich auf dem Development Blog aufgerufen werden.[74]

6.1.1 Programmierung und Methodik

Die Teilstücke (Chunks) der Landschaft sind prozedural generiert. Im Wesentlichen erstellt ein Algorithmus eine zweidimensionale Noisemap (siehe Kapitel 2.3.3.2), welche anhand der Farbabstufung in Höhenpunkte und schliesslich in einem dreidimensionalen Raum übertragen werden kann. Werte wie Dimension, Höhe, Schroffheit (Lakunarität[75]), Detailstufe, etc. des Terrains, können vom Designer innerhalb des Unity-Editors[76] manipuliert werden. Somit fällt diese Variante in die Sparte der human-agent Proc.Gen. Die Farbgebung des Materials des Terrains, ist über ein separates Software-Modul zur Steuerung von 3D-Bildgenerierungsverfahren (Shader[77]) geregelt. Dieser lässt den Designer entscheiden, ab welcher Höhe welche Farbe genutzt und wie stark diese untereinander, bzw. mit einer Textur vermischt werden. Dazu mehr in Kapitel 6.4.

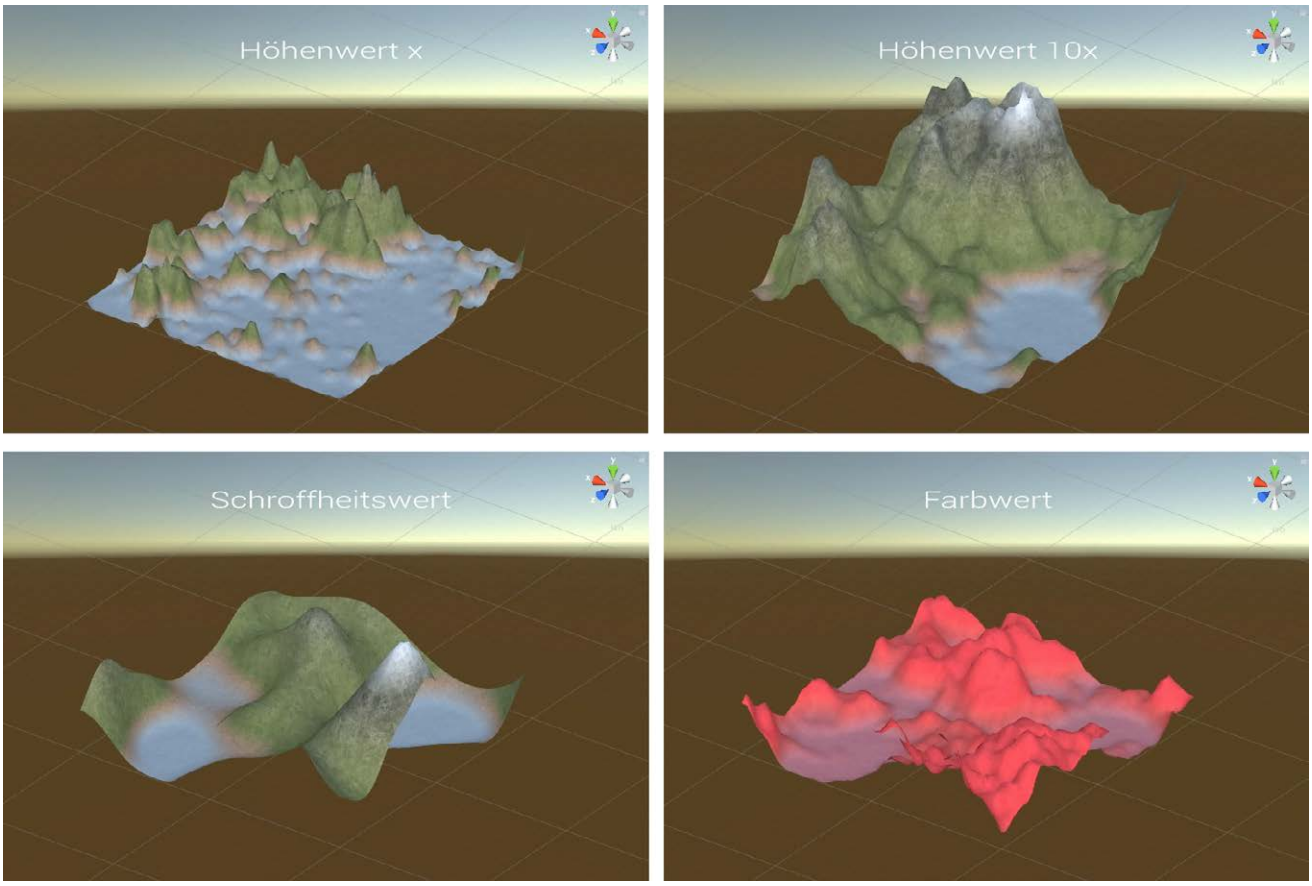


Bild 56: Durch Manipulation von Werten erzeugte varianten eines Terrains

[74] Nenad, Mihajlo: Start building the Landscape Generator (2017) www.mnenad.com/terraintypes-and-essential-first-steps/ (31.05.2018)

[75] Lakunarität (im Kontext des Prototypens) = Abstände zwischen den höchstwwwgelegenen Punkten des Terrains

[76] Editor = Bearbeitungsprogramm. Wikipedia. Begriff: Editor (Software) https://de.wikipedia.org/wiki/Editor_(Software) (15.05.2018)

[77] Shader = Recheneinheiten, die programmiert werden um 3D-Grafiken zu erstellen. Wikipedia. Begriff: Shader https://de.wikipedia.org/wiki/Shader (06.06.2018)

6.1.2 Auswirkungen auf Terrain-Design

Durch die Nutzung besagter Techniken entsteht eine natürlich anmutende, hügelige bis bergige Landschaftsform. Natürlich ist dies abhängig von den manuell eingegebenen Werten, welche aber auf ihre Auswirkung auf die Ästhetik im Editor leicht überprüfbar sind und somit eine einfache Regulierung ermöglichen. Da im Projekt nicht mit Voxeln (siehe Kapitel 2.3.3.2), sondern ausschliesslich mit der digitalen Interpretation von Heightmaps durch den Computer gearbeitet wird, sind bei diesem prozeduralen Schritt keine Höhlenstrukturen möglich. Auch mathematische Gegebenheiten, die die einzelnen Terrain-Teile nahtlos zusammenfügen, bedingen eine Reduktion der Anzahl Detailstufen, in denen die Landschaft darstellbar ist.

Für das Farbdesign der Landschaft ergeben sich durch die komplett manuelle Manipulation kaum Einschränkungen. Der Designer kann unter anderem die Textur, den Farbton und die Überblendungsstäke zwischen den Segmenten nach Belieben verändern. Im Vergleich zum manuellen Designprozess von Terrains in Unity aus Kapitel 2.1.1 ergeben sich jedoch Limitierungen in der Designfreiheit, welche dem prozeduralen Generierungsprozess während des Spielens (vergleiche Kapitel 2.2), geschuldet sind:

- Das Terrain kann nur pro Höhenabstufung designt werden
- Variationen innerhalb einer Höhenstufe sind nicht möglich

Diese Einschränkungen sind im Kontext der Arbeit zu betrachten und sind bei einer Auseinandersetzung in grösserem Zeitrahmen durchaus lösbar. Ich bewerte die Methode dennoch als sehr erfolgreich, da der manuelle Designprozess der Landschafttexturen in einem Endlos-Landschaftsgenerator, wie ihn *GenoTerra* darstellt, weder sinnvoll, noch möglich ist.

6.2 Vegetationsgenerierung

Für die Vegetationsgenerierung kommt human-agent Proc.Gen zum Einsatz. Die Ausgangsformen für jegliche Vegetation in *GenoTerra* sind mit dem Shape-Generator-Tool (Kapitel 5) entstanden, die in einem zweiten Schritt manuell bis zum finalen Asset (Spielbestandteil) ausgearbeitet wurden.

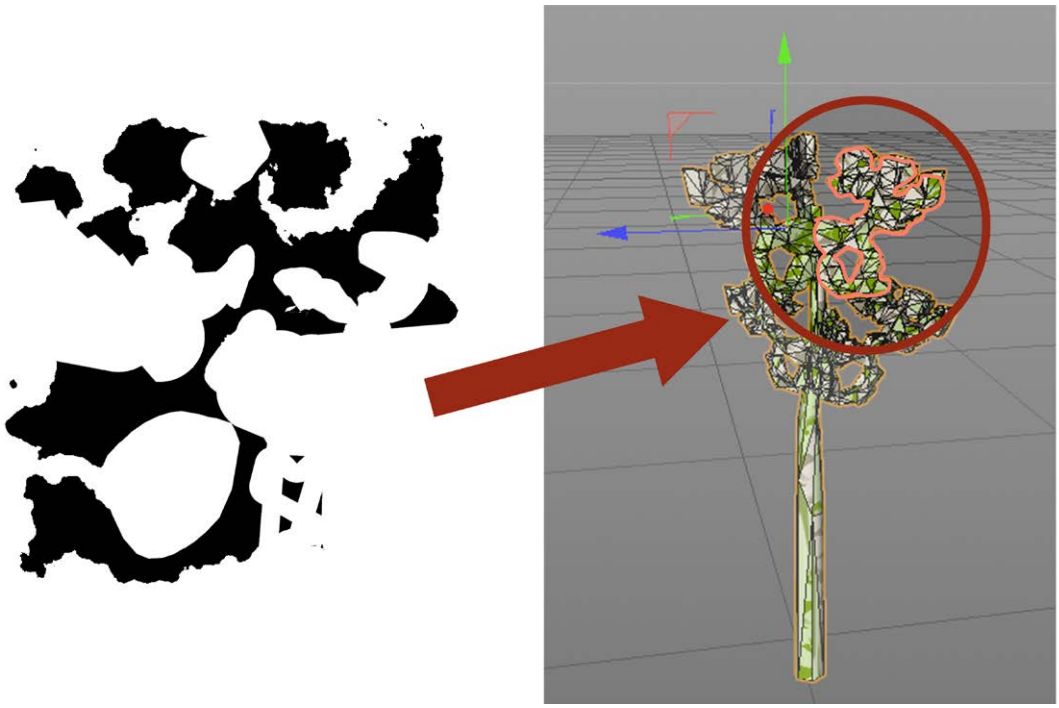


Bild 57: Veranschaulichung einer Verwendungsvariante eines Shapes für Vegetation im 3D-Modell

Dieser Ansatz entwickelte sich erst, als sich die bis dato genutzten, anderen no-agent Proc.Gen Methoden, im Vergleich zu den restlichen Inhalten des Spiels, als zu wenig kohärent in ihrer Formensprache herausstellten. Besagte andere no-agent Proc.Gen Methoden sind primär Varianten von L-Systemen[78], welche zur Generierung von Pflanzenformen eingesetzt wurden.

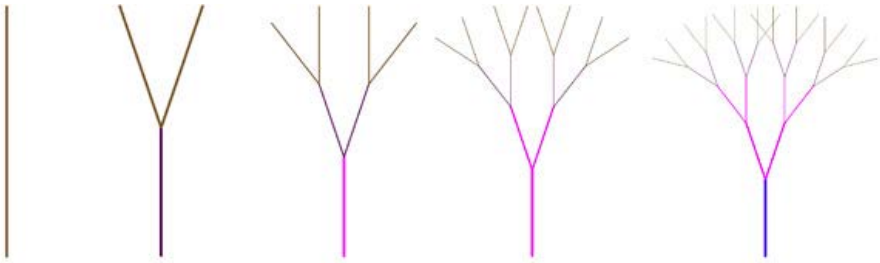


Bild 58: Generierungsschritte eines selbstreferenzieller L-System-Baums

6.2.1 Programmierung und Methodik des Objekt-platzierungssystems für das Terrain

Eine sehr ausführliche Beschreibung der Methodik mit Codebeispielen kann im Development Blog nachgelesen werden.[79] Hier werden drei Varianten von Platzierungssystemen für Vegetation, aber auch andere Landschaftselemente, wie Felsen, Korallen als auch der Platzierung der nicht statisch platzierten Kreaturen in der Welt vorgesellt.

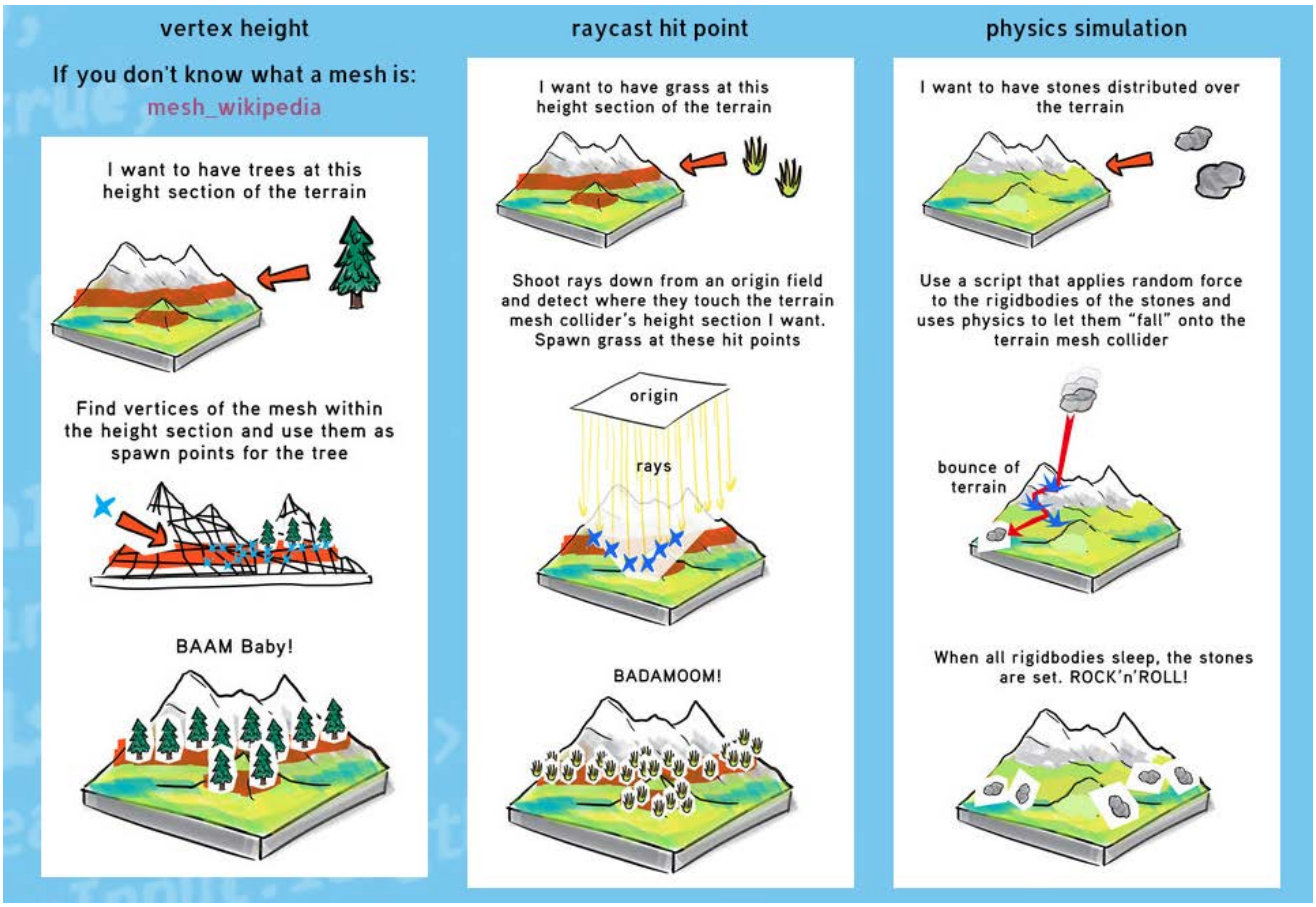


Bild 59: Erläuternder Blogeintrag (Screenshot) mit den drei Varianten zur Platzierung von Vegetation auf dem Terrain

[78] L-System (Lindenmayer-System) sind mathematische Berechnungen, die für selbstreferenzielle Formen (Fraktale) in der Computergrafik genutzt werden können.

[79] Nenad, Mihajlo: Start building the Landscape Generator (2017) www.mnenad.com/objectplacement/ (31.05.2018)

Die Methoden aus Bild 59 sind:

1. Höhenabhängige Platzierung der Objekte auf den 3D-Mesh-Punkten des Terrains
2. Höhenabhängige Platzierung durch Oberflächen-Scan des Terrains
3. Platzierung durch Nutzung des Physik-Systems

Während sich die erste und zweite Methode für alle Objekte eignen, ist die dritte Methode durch die Rotation des am Terrain abprallenden 3D-Objektes, für Vegetation (die in der Regel aufrecht steht) nicht optimal.

6.2.2 Auswirkung auf Inhalte (Vegetation, Objekte)

Einer der wichtigsten Vorteile bei prozeduralen Anwendungen ist der in Kapitel 1.6 be-chriebene Zeitfaktor. Dank dieser no-agent Proc.Gen können in GenoTerra extrem schnell grosse Mengen an Objekten in der Spielwelt platziert werden. Als Nachteil wird die fehlende Feinjustierung der Objekte auf dem Terrain gesehen, da lediglich Start-, und Endhöhen für mögliche Platzierungspunkte der Objekte vom Designer definiert werden. Bei der dritten Variante aus Kapitel 6.2.1 ist es sogar ganz der Software überlassen, wo bspw. der Felsen zum Stehen kommt. Dies simuliert allerdings auch eine gewisse, positiv gewertete Natürlichkeit in der Distribution der Elemente auf dem Terrain.

6.3 Kreaturen-, und Faunadesign

as Designen der Kreaturen im Game beginnt wie das der Vegetation mit der, im Shape-Generator-Tool (Kapitel 5) prozedural generierten Basisform, die anschliessend von Designern zuerst als Konzeptzeichnung (Concept Art), und anschliessend in einer 3D-Modelliersoftware nachgebaut, animiert und texturiert wird.

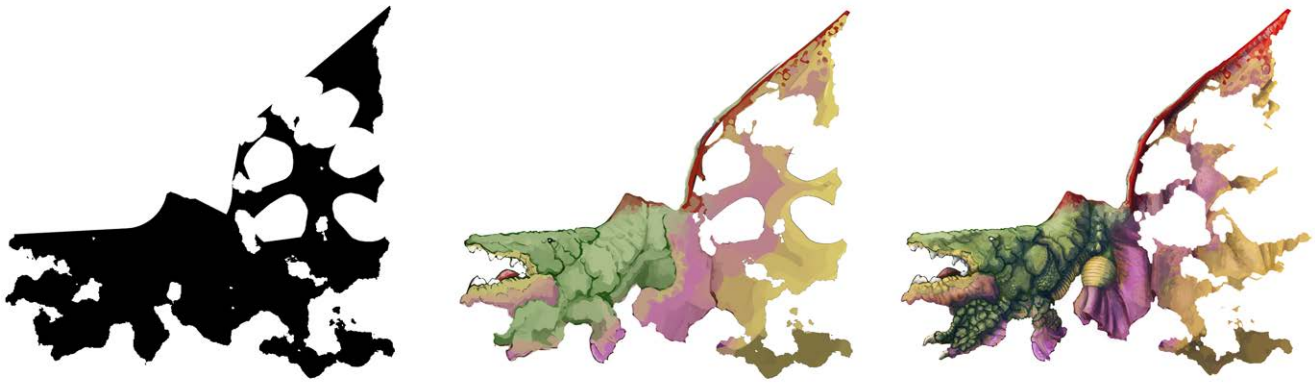


Bild 60: Aufbau eines Kreaturenkonzepts mit einer Ausgangsform des Shape Generators

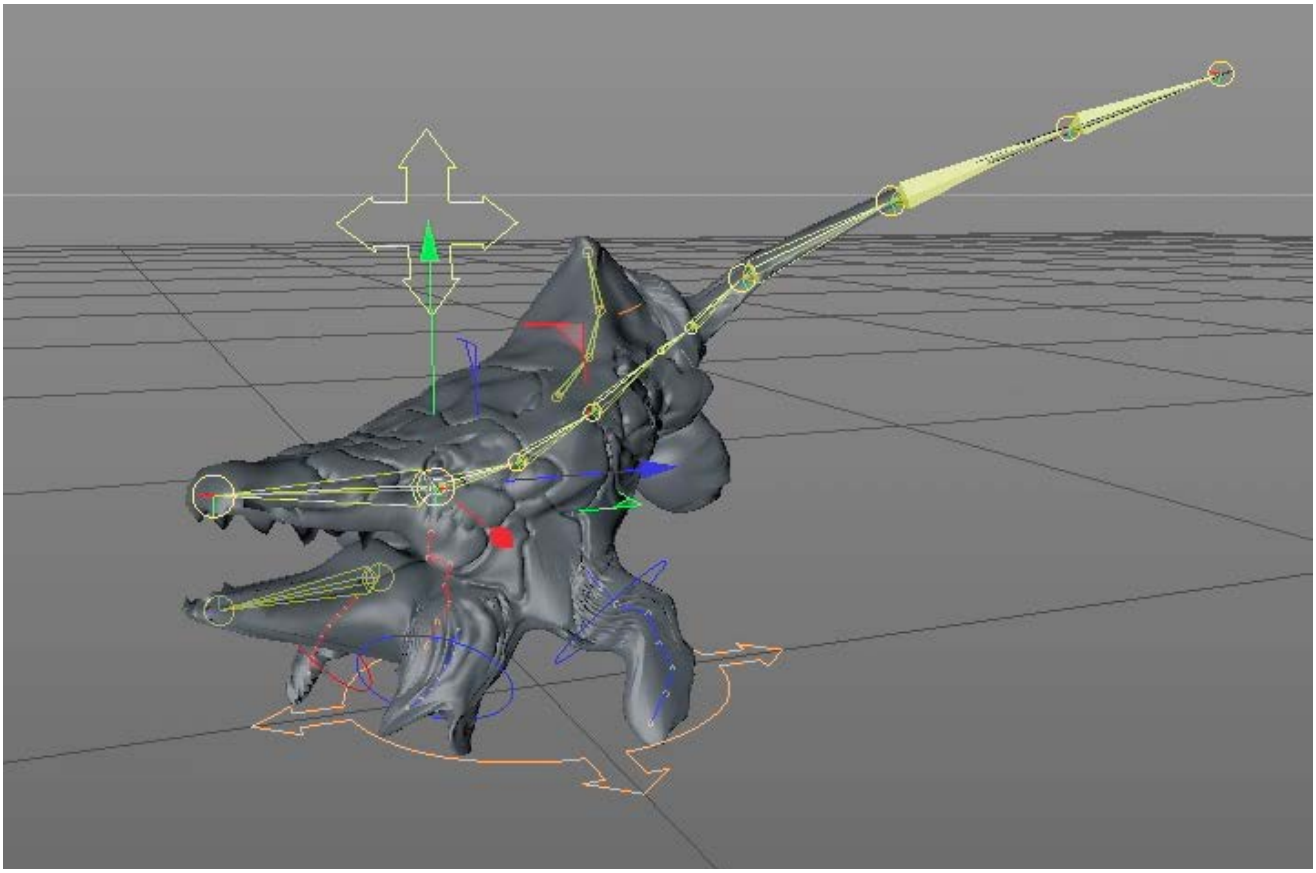


Bild 61: Umsetzung als 3D-Modell als Vorbereitung zur Animierung und Texturierung

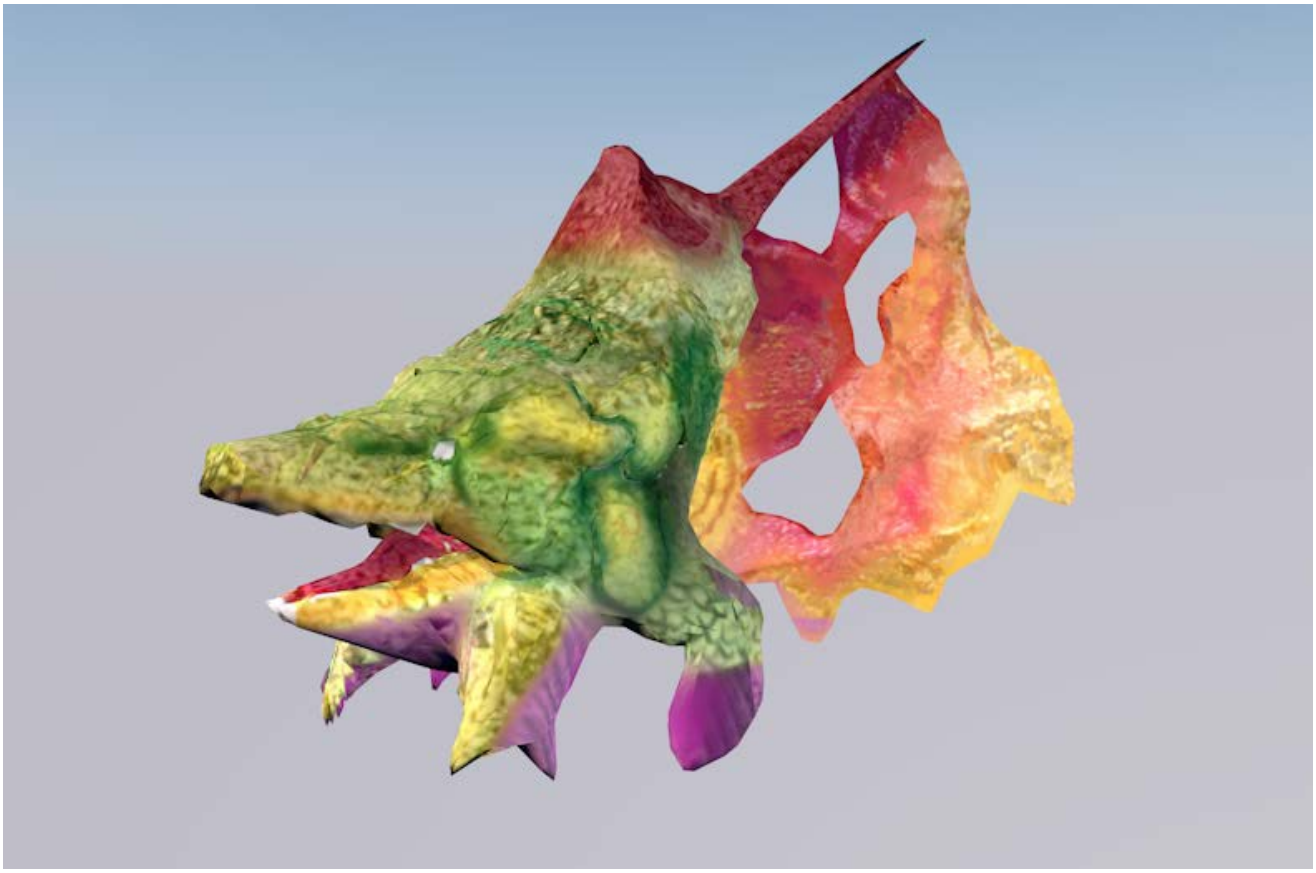


Bild 62: Fertig texturierte Kreatur, wie sie im Game zum Einsatz kommt

6.3.1 Auswirkung auf Kreaturendesign

Dem Designer sind beim Gestalten der Kreaturen grosse Freiheiten bei der Interpretation der Shapes und der 3D-Modellierung gegeben. Die menschliche Kreativität soll bei human-agent Proc.Gen trotz der algorithmisierten Ausgangslage (Shape des Shape-Generator-Tools) essentiellster Bestandteil bleiben. Persönlich eingeschätzt, erleichtert der Einsatz der prozeduralen Ausgangslage (Shape) den Designprozess. Die sonst zur Erstellung erster Skizzen benötigte Zeit kann effizienter genutzt werden (dazu mehr in Kapitel 8.2).

6.4 Texturanwendung und Auswirkung auf das Gesamtdesign der Spielewelt

Die Texturen stammen ebenfalls aus dem oft erwähnten Shape-Generator und werden wo immer möglich an den Spielelementen (Terrain, Spieleravatar, Vegetation, etc.) angewendet.

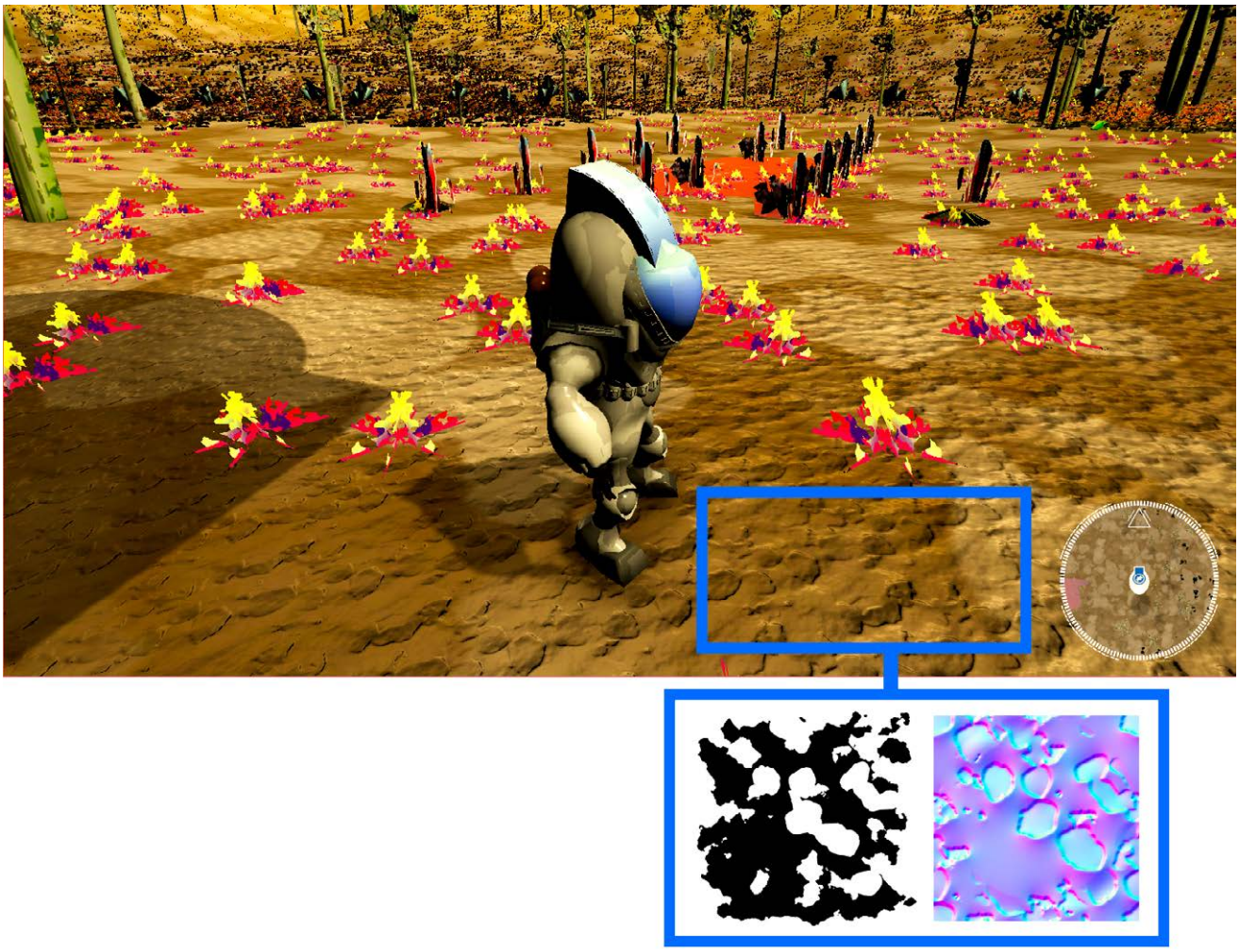


Bild 63: Shape als Textur, als auch als Normal-Map (für optische Tiefe) angewendet auf Boden

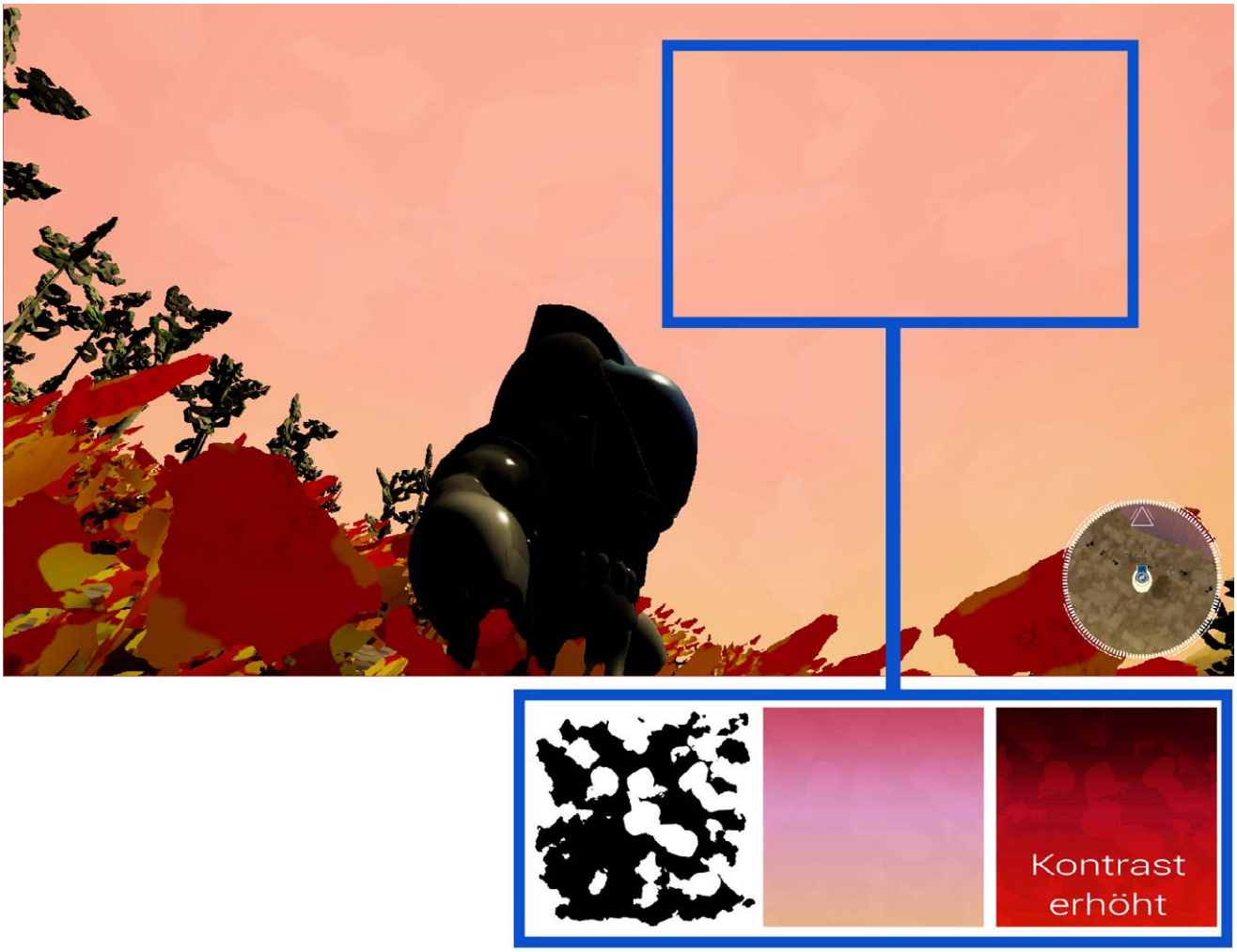


Bild 64: Selbes Prinzip angewandt auf den Himmel (zur besseren Sichtbarkeit wurde der Kontrast im Bild erhöht)

Bild 65: : Weitere Elemente, bei der die Textur Anwendung findet sind: Wellenstruktur von Flüssigkeiten (oben links), Textur des Avatars (oben rechts), Vegetationstextur (unten links), Textur der Kreaturen (unten rechts)

Diese selbsterdachte Anwendungstechnik soll zu einer Verbesserung der visuellen Kohärenz der Spielwelt führen, indem sich die einzelnen Spielelemente in ihrer Textur, bzw. Musterung optisch angleichen.

6.5 Entwicklung der Story

Initialidee für die Story, also der Narration des Games, war eine Auseinandersetzung mit realen, ökologischen Themen. Der Spieler sollte durch das Spielen auf die Veränderung von Landschaftsformen durch den Klimawandel sensibilisiert werden.

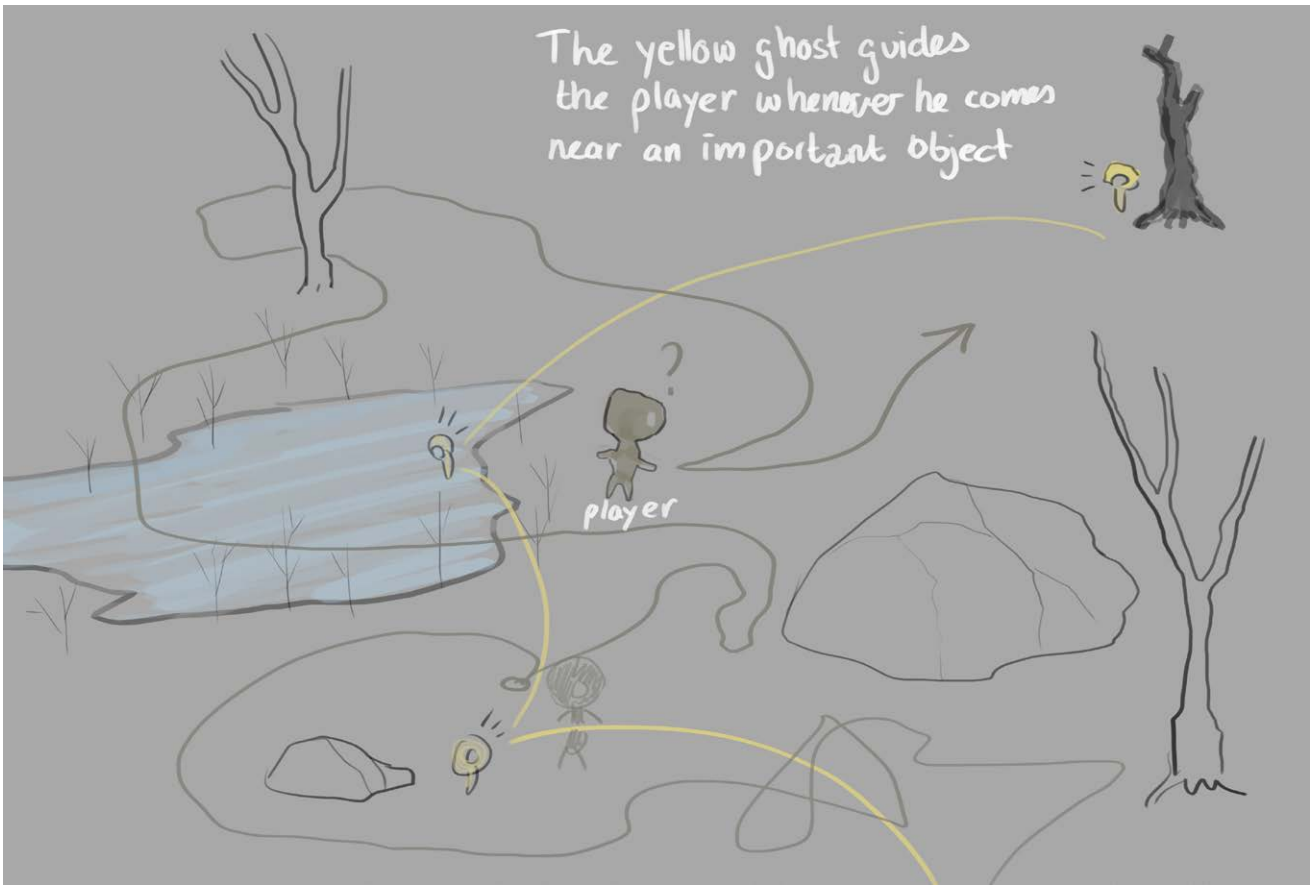


Bild 66: Frühes Konzept für „Helferchen". Diese sollten dem Spieler die, durch unökologisches Handeln der Menschen zerstörte Landschaft zeigen und den Spieler in der offenen Spielwelt leiten. Diese Idee wurde in Teilen in der sprechenden Drohne von GenoTerra umgesetzt

Bild 67: Links: Drohne (Farblich hervorgehoben) leitet ein Tutorial, bei dem der Spieler die Steuerung des Avatars kennenlernt. Rechts: Drohne umkreist den Spieler und gibt hilfreiche Informationen, als auch zufällige oder witzige Kommentare ab.

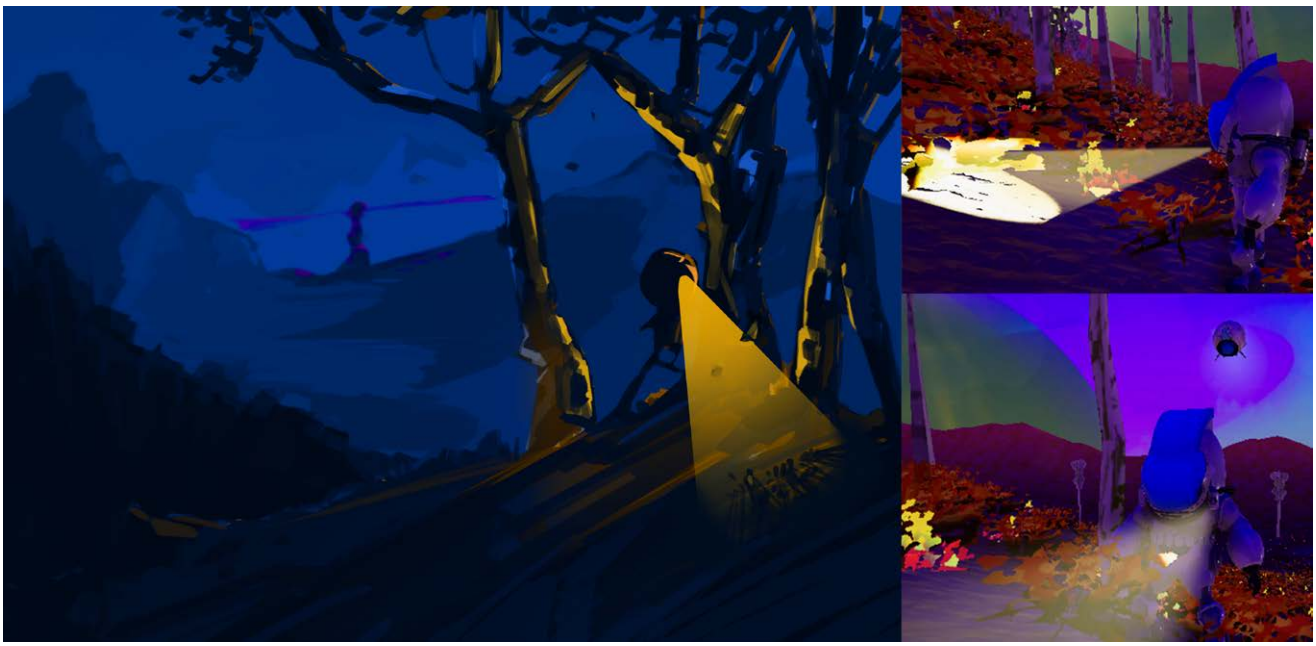


Bild 68: Frühe Konzeptskizze. Spieler orientiert sich in einer finsteren Landschaft mit einem Licht. Der Aspekt des nächtlichen Erkundens wurde in den Prototypen integriert.

Die Idee einer ökologischen Thematik wich mit der Rückbesinnung auf die Kernkompetenzen eines Designers und der Konzentration auf die neu entwickelte Leithypothese (Kapitel 1.8). Zudem ist die schiere Menge an Rohdaten, welche für eine seriöse Bearbeitung einer ökologischen Fragestellung noch aufbereitet werden müsste, im Rahmen einer Masterthesis kaum zu bewältigen. Die ökologische Thematik an sich ist aber durchaus ein Feld, dem ich mich weiterführend gerne widmen möchte (Siehe Kapitel 7.3).

Die Story soll dem Designer die Möglichkeit bieten, eine Spielwelt auf ihre visuelle Kohärenz hin zu bearbeiten. Um dies zu ermöglichen wurde als Szenerie ein erdähnlicher Planet gewählt, um ein Science-Fiction-Setting zu ermöglichen. Beim Designen dieser Planetenumgebung können physikalische Gegebenheiten ausser Acht gelassen und biologisch „unmögliche“ Wesen, sprich Vegetation-, und Landschaftsformen kreiert werden. Der Anspruch liegt im Rahmen dieser Arbeit ausschliesslich in der visuellen Bildsprachenerzeugung. Die komplette Story kann in der Dokumentation (Kapitel 10.2) nachgelesen werden.

Bild 69: Raumschiff in dem sich der Avatar befindet über Kepler-69c, dem erdähnlichen Exoplaneten, der auch die Textur einer generierten Form (Shape) aufweist

Die Bilder 69 bis 71 sind Teile des Intros, welches den Spieler in die Story einführen. Sie entstanden als direkte Folge der Auswertung der Playtestings (Kapitel 7.2). Diese machten klar, dass ohne die Story das Spielziel, und damit die Motivation für einige Spieler, die spielenden Personen nicht erreicht.

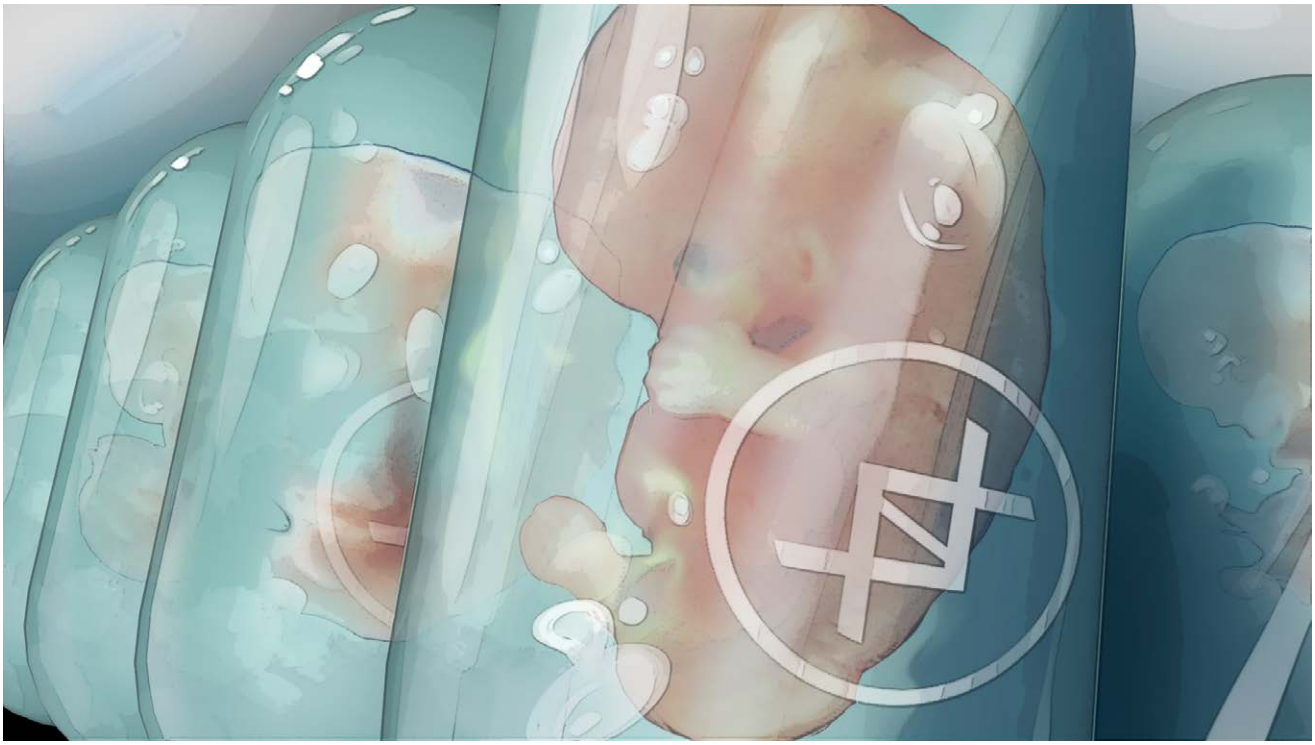


Bild 70: Menschliche Embrionen werden in Behältern ausserhalb der unbewohnbar gewordenen Erde aufbewahrt. Das dargestellte Logo (DNA-Doppelhelix-Design in Kreisform) kommt in GenoTerra an diversen Stellen vor (Avatar-Raumanzug, Drohne, grafische Elemente des User Interfaces, etc.) und steht für die Weltraummission des Spielers

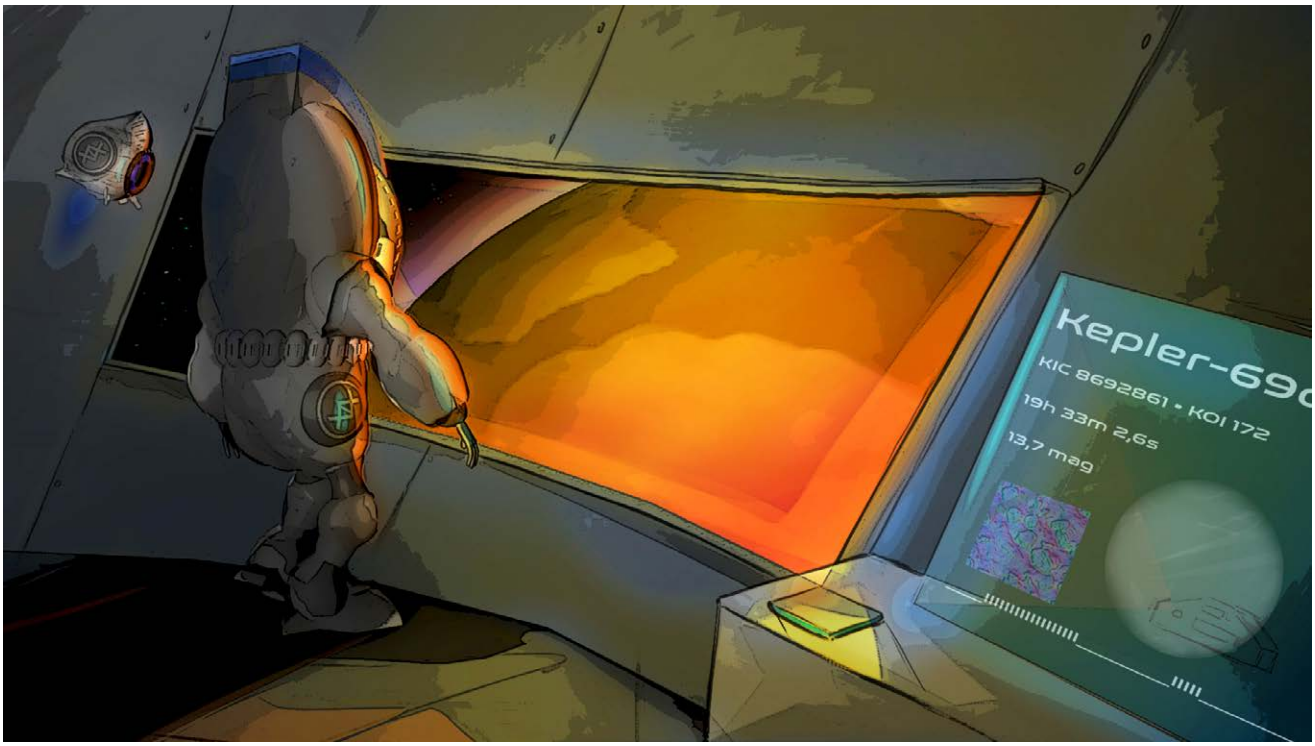


Bild 71: Avatar des Spielers und Drohne blicken kurz vor der Landung auf Kepler-69c (eigentlicher Spielstart) aus dem Raumschiff

Stilistisch sind diese selbsterstellten Illustrationen nicht direkt auf die Ästhetik des interaktiven Games selbst übertragbar. Es ist bei Gameproduktionen aber durchaus üblich Illustrationen (bei Titelbildern, Werbeaktionen, etc.) und die effektiven, visuellen Spielinhalte getrennt zu behandeln. Die Entscheidung, mich eines comichaften Stiles zu bedienen, beruht auf persönlicher Präferenz und Kompetenz in diesem Feld.

Zusätzlich zu den bereits umgesetzten Aspekten, ergibt sich ausserdem die Möglichkeit, die Shapes in Zukunft zu nutzen, um die Spannungsbögen der Narration zu definieren.

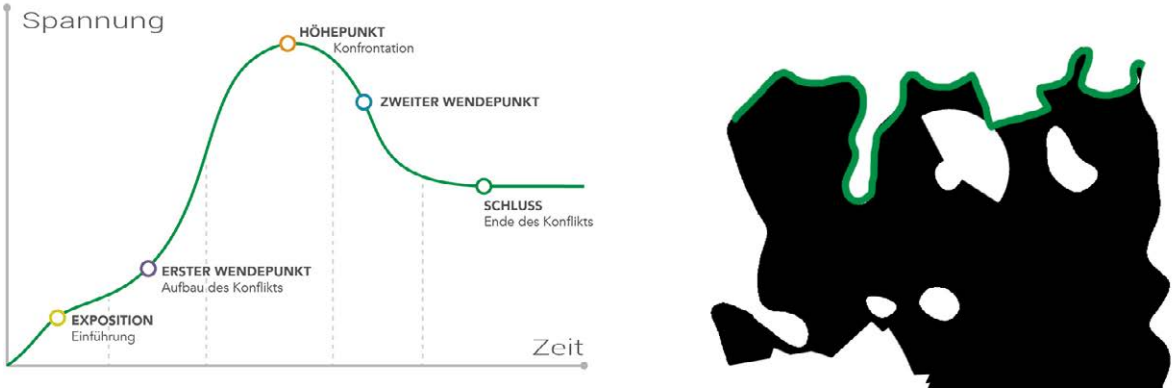


Bild 72: Eine simple Spannungskurve (links) kann auch einer generierten Form entnommen werden (rechts), wobei die kreative Leistung des Erdenkens der Geschichte hier beim Menschen liegen würde

Denkbar ist auch, die Narration mit der Entfernung des Spielers zum Startpunkt aufzubauen. Dies wäre logisch, da sich die Fähigkeiten des Avatars verbessern während der Spieler in der Welt voranschreitet.

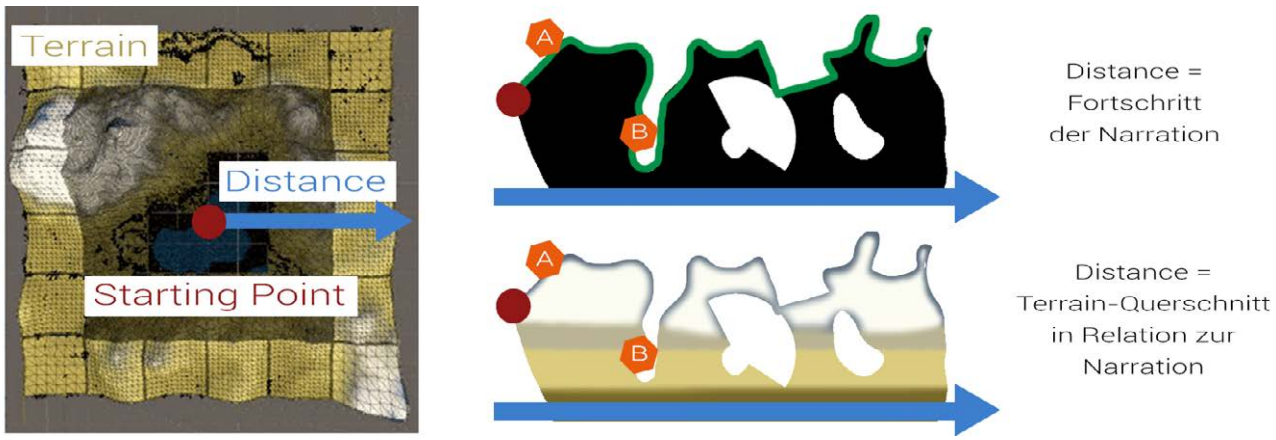


Bild 73: Terrain-Teilstücke aus der Vogelperspektive betrachtet. Der blaue Pfeil zeigt die Distanz (engl. Distance) in eine Richtung an. Dieser bildet in den beiden anderen Abbildungen zugleich den Narrationsfortschritt.

Um obiges Schema besser zu verstehen, hier ein Beispiel:

Situation **A** aus Bild 73 könnte das abenteuerliche Erklimmen eines Berges im Querschnitt darstellen. Die Spannung steigt, da man nicht weiss, was sich auf dem Gipfel befindet. In Situation **B** befindet sich der Spieler fast schon am Grund eines Canyons. Es droht keine Gefahr und man erwartet einen wiederholten Aufstieg (Spannung sehr gering). All das spielt sich ab, während man sich als Spieler vom Startpunkt (Starting Point) entfernt und immer schwierigere landschaftliche Herausforderungen mit neuen Spannungsmomenten erlebt.

6.6 Entwicklung der spezifischen Spielmechanik

Für diese Arbeit ist Spielmechanik definiert, als das Zusammenspiel vom einprogrammierten Regelwerk und den Bedingungen zur Erfüllung des Spielziels, ohne welches das Game ein Play wäre (vergleiche Kapitel 1.1.1).

> Einprogrammiertes Regelwerk (Spieler kann hüpfen)
>
> Bedingung für Erfüllung des Spielziels (Spieler muss über alle Hindernisse hüpfen) = Spielmechanik

Um eine Spielmechanik zu finden, die Spass bringen kann, ist das unablässige Testen der eigenen Ideen mit Drittpersonen ohne Bezug zum Projekt ein sehr wertvolles Hilfsmittel. Bei *GenoTerra* wurde beim Erstellen eines Papier-Prototyps bereits klar, dass die Spielmechanik in der Thesis eine untergeordnete Rolle spielen würde, da es primär um die Beantwortung der Forschungsthese (vergleiche Kapitel 1.8) geht. Zur Findung einer passenden Spielmechanik, die den Spieler das Betrachten der Spielwelt aus zwei Blickpositionen (Third-Person-View und First-Person-View) ermöglichen würde, erschien mir eine „Fotografier-Mechanik" als geeignete Lösung.

Bild 74: Oben: Im Third-Person-View sieht der Spieler seinen Avatar von aussen und kann die „Kamera" meist um ihn herum rotieren | Unten: Im 1st-Person-View sieht der Spieler die Welt aus der Ich-Perspektive des Avatars. In GenoTerra sieht der Spieler im Scan-Modus, zusätzlich eine Benutzeroberfläche mit grafischen Elementen und Informationen

Ein wichtiger Schritt zur Entwicklung einer passenden Mechanik war zudem zu realisieren, wie der Blick des Designers eines Games sich von dem, des Spielers unterscheidet. Während nämlich der Designer das Game vom Kern aus betrachtet, also quasi von der Programmierung zur Ästhetik, nimmt die spielende Person das Game genau anders herum wahr.[81]

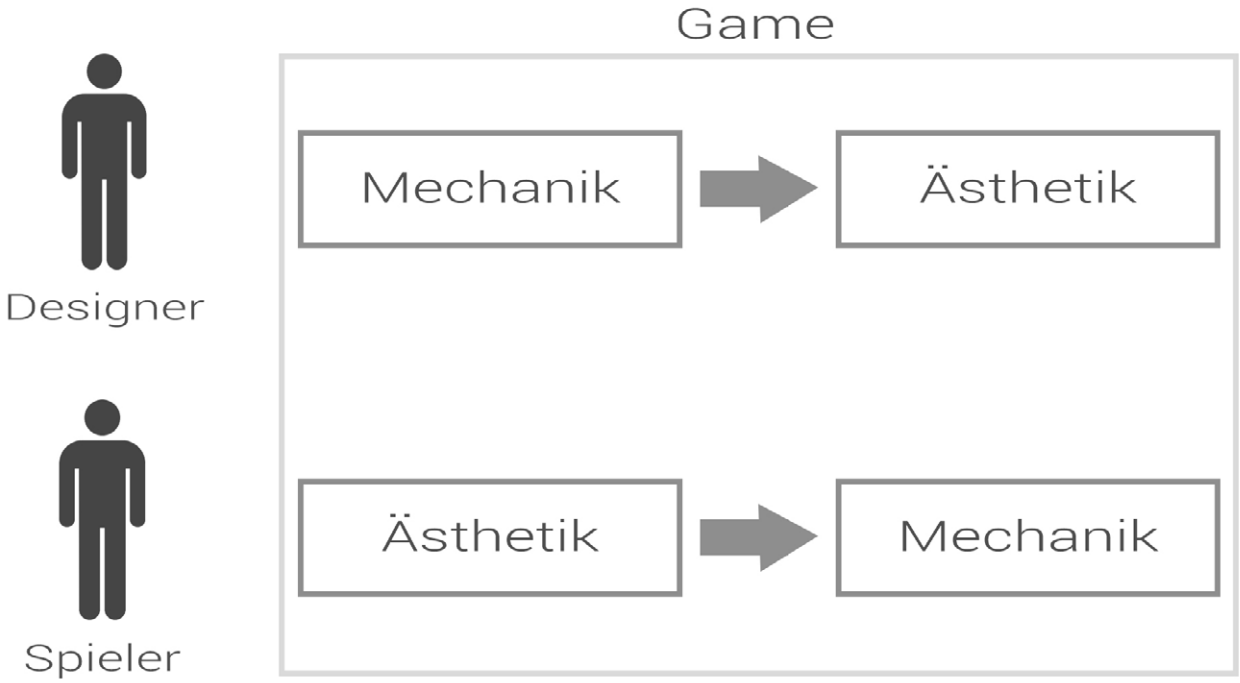


Bild 75: Wahrnehmung eines Games vom Gamedesigner selbst (oben) und vom Spieler des Games (unten)

Anstelle einer „Fotomechanik" entschied ich mich schliesslich für eine, der Story angepasste „Scan-Mechanik".

Mechanik bei *GenoTerra*: Spieler muss Alien-Wesen in der Spielwelt finden, diese im Scan-Modus (siehe Bild 76) für eine bestimmte Zeit und in genügend geringem Abstand fokussieren um ihre DNA zu scannen

Spielziel bei *GenoTerra*: Genügend Individuen pro Alien-Spezies einscannen um alle Balken zu füllen und Menscheit zu retten

[8] Rehfeld, Gunter, Schmidt, Ulrich (Hrsg.): Game Design und Produktion. München. Hanser Verlag (2014) S.1

Bild 76 : Spieler hat eine Alien-Kreatur fokussiert und deren DNA vollständig gescannt

Bild 77: Im Bestiarium kann der Spieler den Füllstand der Balken per Spezies sehen und erkennt, wie viele Individuen er/sie noch scannen muss. Für die Erkennungsbilder wurden die Ursprungs-Shapes der Kreaturen wiederverwendet

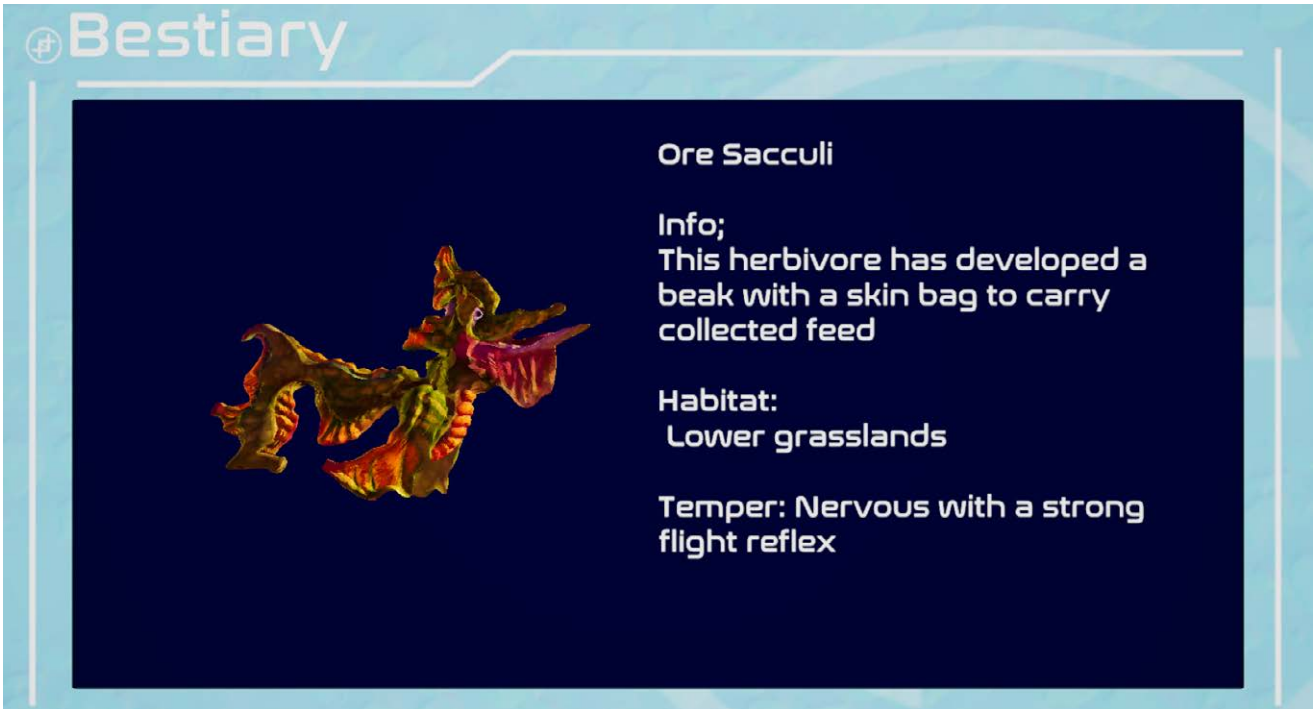


Bild 78 : Alle Kreaturen, dessen DNA der Spieler im Game bereits einmal erfolgreich einscannen konnte, sind mit Zusatzinformationen und einer 360°-Ansicht des Aliens im Bestiarium verzeichnet

6.6.1 Ähnliche Spielmechaniken in bestehenden Games

Als guter Startpunkt zur Ausarbeitung einer Fotografier-, bzw. Scan-Mechanik schien eine Analyse bestehender Spiele, die ähnliche Mechaniken nutzen. Folgend ist eine Auswahl dieser dargestellt und im Entwicklungskontext des eigenen Prototyps kommentiert.

6.6.1.1 Pokémon Snap

Das im Jahre 2000 in Europa veröffentlichte und unter dem Genre „Fotosafari" gelistete Game *Pokémon Snap* [16] nutzt das Fotografieren als Spielmechanik.[82] Die Story des Spiels dreht sich um einen jugendlichen Avatar, der von einem „Professor Eich" genannten Wissenschaftler den Auftrag erhält, die Pokémons zu Forschungszwecken aus einem Schienenfahrzeug her zu fotografieren. Für eine Auswahl der Fotos der Pokémon, erhält der Spieler, am Ende der zeitlich begrenzten Safari auf einem vorgegebenen Pfad, Punkte. Die Höhe der Punktzahl hängt davon ab, wie gut oder in welcher Pose die Pokémon auf den Schnappschüssen zu sehen sind.

Persönlich als positiv bewertet werden:
- Eingliederung und Konzentration auf eine Mechanik (das Fotografieren)
- Punktevergabesystem am Ende

Persönlich als negativ bewertet werden:
- Eingeschränkte Bewegungsfreiheit
- Aus heutiger Sicht geringe Anzahl an nur sieben Leveln (in sich geschlossene Spielwelten)

[82] https://www.pokewiki.de/Pokémon_Snap (16.05.2018)

Bild 79 : Spieler lernt, wie man Fotos der Pokémon macht

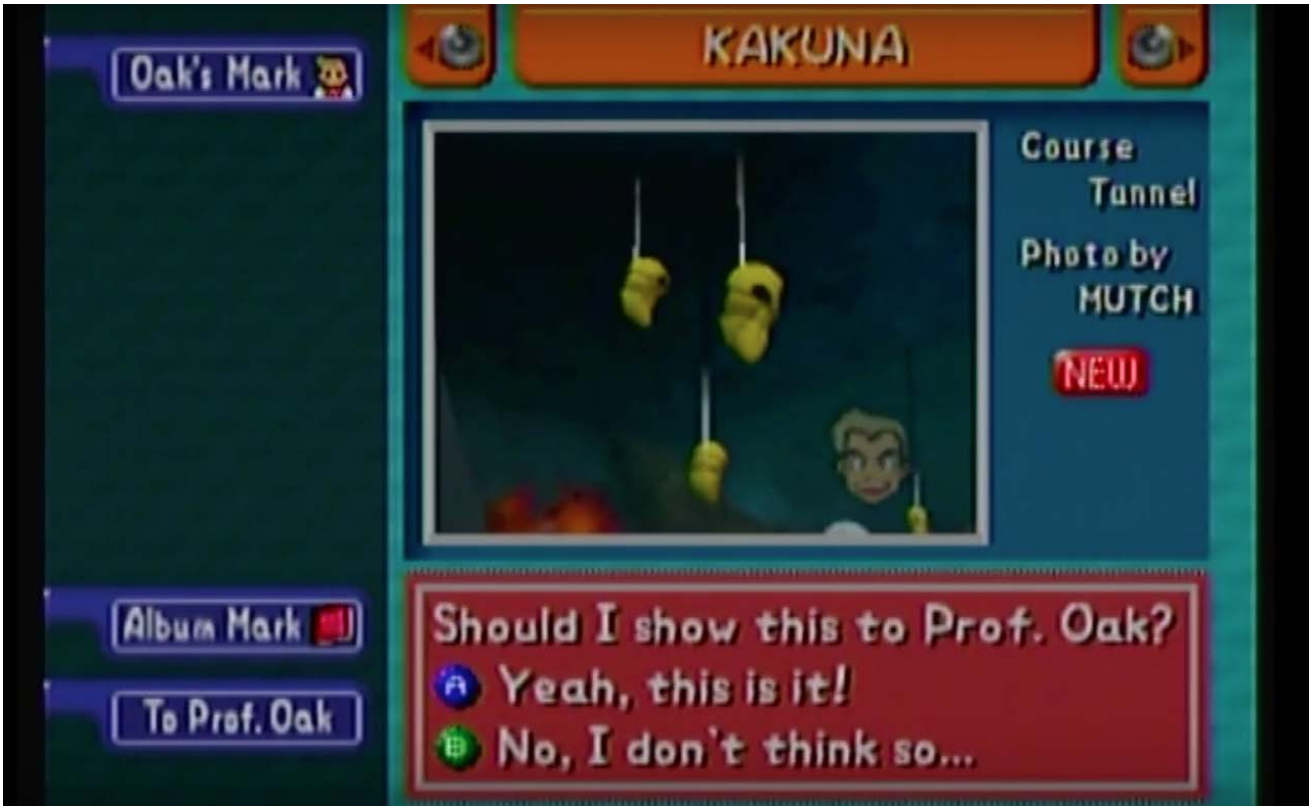


Bild 80 : Spieler entscheidet, welche Fotos er dem Professor zur Bewertung geben möchte

6.6.1.2 Beyond Good and Evil

Im 2003 erschienenen Game *Beyond Good and Evil* [17] übernimmt der Spieler die Rolle einer Heldin namens Jade. Jade muss auf einem, von Aliens kontrollierten Planeten Geld für das Aufrechterhalten des Schutzschildes gegen die Aliens verdienen. Sie tut dies mit einem Job als Fotografin.[83]

> „When Jade runs out of money to run the shield that protects them, she finds a photography job, cataloguing all the species on Hillys for a science museum." [84]

Bild 81 : Das Halten vom Fokus auf die Kreaturen gehört zur Mechanik

Für das Fotografieren jeder neuen Spezies erhält der Spieler eine gewisse Summe Geld, abhängig davon, wie selten-, bzw. wie schwierig die Wesen zu finden sind.

Persönlich als positiv bewertet werden:

- Freie Bewegungsmöglichkeit und damit verbundenes Suchen nach den Wesen
- Einfachheit des Punktesystems

Persönlich als negativ bewertet werden:

- Spielunterbruch nach erfolgreicher Nutzung des Fotoapparates, um Punktemenü aufzurufen und Belohnung (Geld) zu vergeben (vergleiche Bild 82)

---

[83] Wikipedia. Begriff: Beyond Good and Evil https://en.wikipedia.org/wiki/Beyond_Good_%26_Evil_(video_game) (16.05.2018)

[84] Ebd.

Bild 82 : Punktesystem für das erfolgreiche schiessen eines Fotos

6.6.1.3 Expedition Perm

Expedition Perm [18] ist die Arbeit von Stephanie Stutz, einer ehemaligen Studentin der ZHdK[85] für das Museum für Naturkunde Chemnitz[86].

> „My goal was to reconstruct the forest and some animal species from the time of Perm and make that world available to the public for the first time." [87]

Persönlich als positiv bewertet werden:

- Auseinandersetzung mit der Thematik des Erdzeitalters des Perm (Kohärenz von Story und Design)

Persönlich als negativ bewertet werden:

- Sehr geringe Interaktionsmöglichkeit im Spiel

[85] Zürcher Hochschule der Künste (ZHdK) https://www.zhdk.ch (16.05.2018)

[86] https://www.naturkundemuseum-chemnitz.de (16.05.2018)

[87] Stutz, Stephanie: Expedition Perm (Jahr unbekannt) https://stephaniestutz.ch/projects/5651361 (16.05.2018)

Bild 83: Der Spieler fotografiert gerade ein urzeitliches Tier des Perm

7. Evaluation des Prototyps zu Bildsprachen-kohärenz

Zur Evaluation des Prototyps *GenoTerra* ist es wichtig zu wissen, dass sie sich auf *GenoTerra* (Stand 05.06.2018) bezieht. Also ein nicht finaler, aber visuell bereits weit entwickelter Prototyp des Games, an welchem sich Spieltests (Playtestings) mit Fragen zur visuellen Kohärenz lohnten. Diese Evaluation stellt quantitativ keine repräsentative, empirische Untersuchung dar, sondern ist vielmehr eine im Rahmen der Thesis sinnvolle, sowie inhaltlich und methodisch fundierte Bestandsanalyse. Alle Antworten sind anonymisiert und wörtlich in der Dokumentation unter Kapitel 10.4 überprüfbar. Ausserdem wurde das Spielverhalten beobachtet und Notizen dazu erstellt.

7.1 Playtesting mit Interviews

Die zwölf Testpersonen sind Laientester (Studenten), die *GenoTerra* auf folgende Punkte hin kommentiert haben:

- **Frage A**: Beschreibe die Welt, in der das Game spielt:
- **Frage B**: Wo siehst, hörst, spürst du Ähnlichkeiten zwischen Spielelementen?
- **Frage C**: Was in der Welt passt visuell nicht hinein?
- **Frage D**: Wo und warum hast du Begeisterung gespürt?
- **Frage E**: Wo und warum hat dich das Spiel gelangweilt?
- **Frage F**: Was in der Spielewelt fühlte sich unecht an? (Im Sinne von „Was zerstört die Immersion[88]“)

[88] Immersion = Im Kontext der virtuellen Realität bezeichnet „Immersion“ den Zustand, in dem der Nutzer das Bewusstsein, sich in einer künstlichen Welt zu befinden, verliert. Er lässt sich mit allen seinen Sinnen auf das Erlebnis ein und kann, im Gegensatz zur filmischen „Immersion“, mit der virtuellen Realität interagieren.
Bockholdt, Nikolai: VR, AR, MR und was ist eigentlich Immersion? (2017) https://www.thinkwithgoogle.com/intl/de-de/marketingkanäle/innovative-technologien/vr-ar-mr-und-was-ist-eigentlich-immersion/ (06.06.2018)

Ziel war es, den Prototypen auf die Kriterien für Bildsprachenkohärenz aus Kapitel 1.6.2 hin zu überprüfen.

Gespielt wurde auf einem Xbox360-Controller. Dies ist insofern erwähnenswert, da der Spieler zu Beginn des Games und anhand der hilfeleistenden Drohne (vergleiche Kapitel 6.5) lernt, seinen Avatar durch diese Schnittstelle (Controller) zu bedienen.

Bild 84 : Testsituation: Testperson spielt GenoTerra mit XBox360-Controller in den Händen

7.2 Validierung der Playtestings

Anhand der dokumentierten Playtestings in Form von Fragebögen (Kapitel 10.4) kann gemutmasst werden, dass die Spielwelt durchaus als erdähnlicher Planet erkannte wurde. Auch die Farbgebung/ Farbstimmung wurde von den Probanden vorwiegend positiv bewertet. Gründe könnten der starke Kontrast einiger Pflanzen sein, die in der etwas dunklen Nachtsituation hervorstechen. Überaschenderweise wurde die Spielwelt von einigen Testpersonen als „klein“ wahrgenommen, obwohl die Welt theoretisch endlos ist und auch so erscheinen soll (vergleiche Kapitel 6.1). Eine Erklärung dafür könnte die Startposition sein, was durch Kontrollfragen zukünftig weiterführend evaluiert werden wird. Diese Startposition des Avatars bei Spielbeginn ist von der einen Seite von Bergen umschlossen und an der anderen von einem orangen Gewässer visuell eingegrenzt.

Bild 85: Links: Startsituation (nachts) mit Blick auf Bergkette im Westen und farblich herausstechender Vegetation. Rechts: Oranges Gewässer gen Osten des Startpunktes

Andere Spieler hingegen waren von den Bergen so angetan, dass sie ohne zu zögern direkt diese ansteuerten um sie zu Erklimmen. Dies kann einerseits als persönlicher Erfolg gedeutet werden, da der Entdeckergeist der Spieler geweckt wurde. Andererseits unterbrachen die Spieler durch das Ignorieren der Instruktionen der Drohne (Drohne erklärt wie man den Avatar steuert) und der unmittelbaren Exploration das Bedienungs-Tutorial und wussten später nicht, was sie tun müssen um in der Geschichte weiter zu gelangen. Dies führte dazu, dass die Spieler zwar durchaus Spass beim Erkunden der Spielwelt hatten, einige wenige jedoch die Kernaufgabe (Scannen der DNA der Kreaturen) ohne Hilfe nicht als solche erkannten. Das Tutorial wurde als verbesserungsbedürftig qualifiziert, da Spieler mit erstaunlicher Mehrheit Befehle wie:

„Drücke jetzt den A-Knopf“

befolgten. Jedoch bei Sätzen wie:

„Du kannst START drücken um herauszufinden wie du rennst“

nicht auf den Text, bzw. das Gesprochene reagierten.

Generell sind bis dato folgende Schlüsse bedeutsam:

- Nachtsituation behindert Orientierung. Hellere, bzw. Tagsituation wäre klarer.
- Instruktionen müssen kurz und als unmissverständliche Aufforderung formuliert sein.
- Es muss ein Leitsystem geben (Wege, Lichter, Signale, etc.), welches die Spieler zu Zielen (Wesen zum Scannen, etc.) führt.

Zu letztem Punkt ist die Mini-Karte (Mini-Map) eine gute Basis, welche aber zum Zeitpunkt der Playtestings nicht weit genug entwickelt, bzw. bei der Orientierung in der Spielwelt nicht hilfreich war. Zwei Verbesserungsvorschläge sind in Bild 87 illustriert:

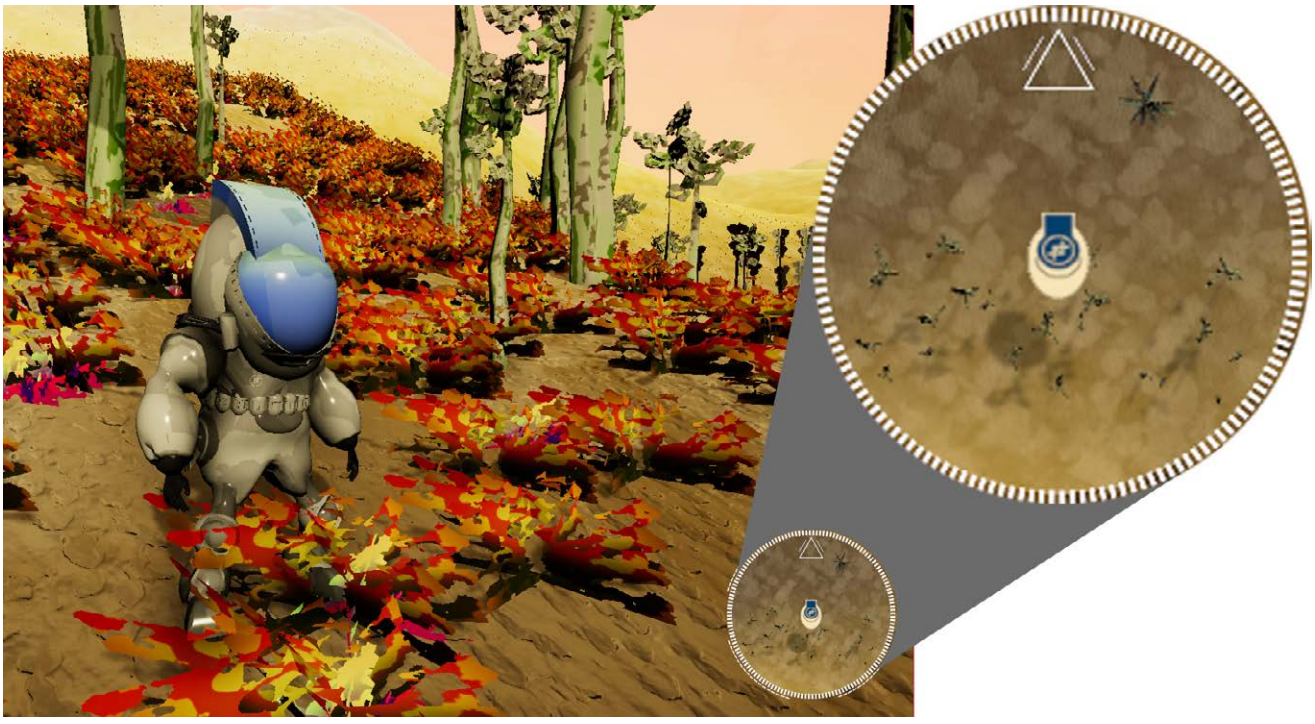


Bild 86: Die Mini-Map bietet dem Spieler in dieser Form noch kaum Vorteile bei der Orientierung

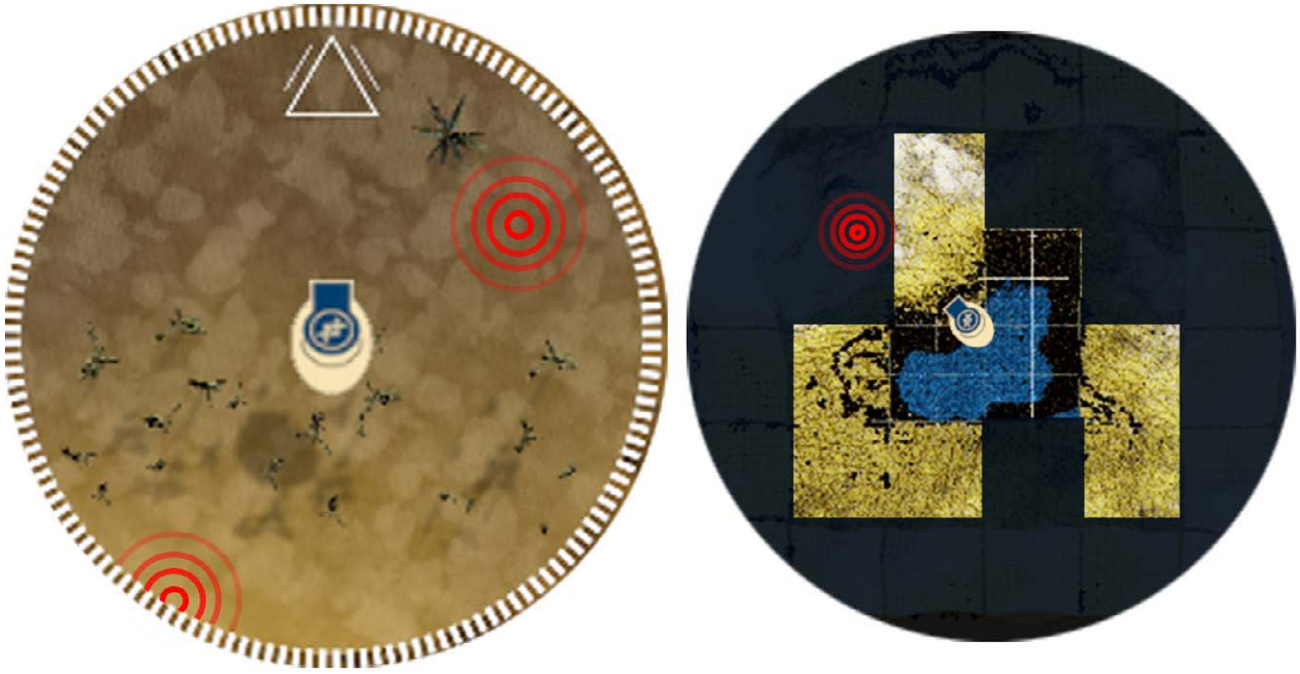


Bild 87: Links: Mini-Map mit Signalen (akustisch, visuell, etc.) an Stellen mit wichtigen Gegenstände oder Wesen in der Spielwelt. Rechts: Mini-Map mit viel grösserem Spielwelt-Ausschnitt und dunklen Bereichen, die noch nie besucht wurden

Aussagekräftig für die parallelen Verhaltensweisen in realen Welten und Spielwelten ist die Beobachtung, wie Testspieler wegen der zu dato fehlenden Orientierungsmöglichkeiten anhand der Mini-Map mit prägnanten Landschaftsformationen als Orientierungshilfen umgingen. So wurde von einigen Testern beispielsweise das Besteigen von sehr hohen Bergen, in der Erwartung vom Gipfel herab die Landschaft überblicken zu können, aktiv ausgeübt. Die Testversion des Prototyps war jedoch auf sogenanntes Clipping (Kamera rendert nur eine gewisse Distanz, alles andere wird nicht gezeigt) angewiesen. Diese Methode zur Verringerung von Rechenaufwand für den Computer, zugunsten des Spielflusses führte dazu, dass genau dieses Panorama gar nicht sichtbar war. Im Gegenzug wurde die sogenannte „Skybox" sichtbar. Die Skybox ist quasi der Himmel, welcher im 3D-Raum als Würfel die Texturen des Himmels trägt.

Bild 88: Spieler ist so hoch hinaufgelangt, dass die Clipping-Distanz nur noch den Gipfel selbst beinhaltet. Der Boden ist abgeschnitten und man kann die Skybox erkennen

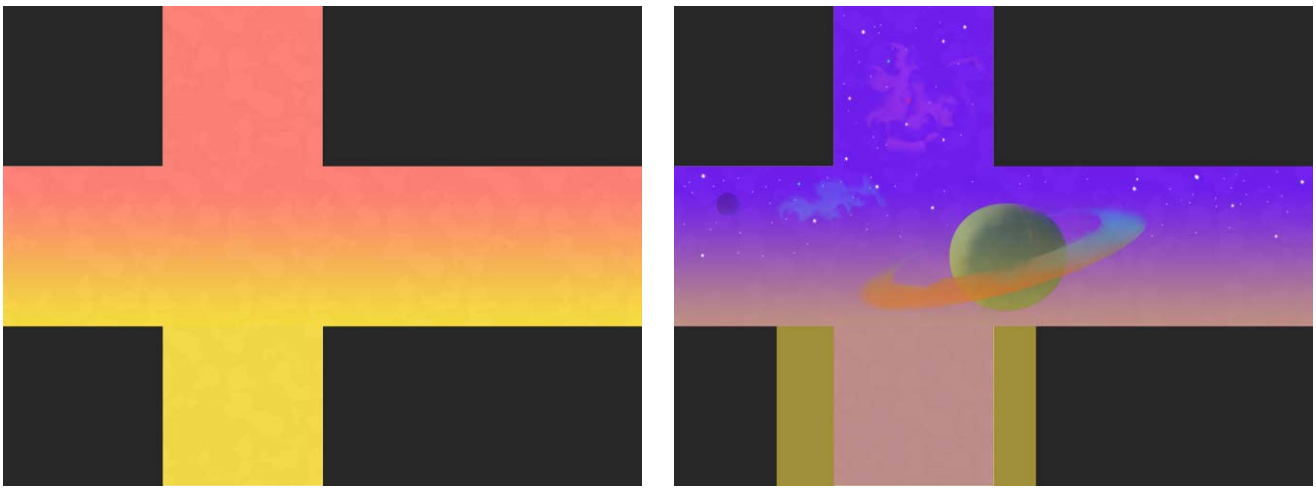


Bild 89: Links: Skybox (Tag). Rechts: Skybox (Nacht). Bei beiden sind die Texturen des Shape Generators angewendet (siehe Kapitel 6.4)

Leider war die Skybox auch in anderen Situationen als Würfel wahrnehmbar. Persönlich war es beim Spielen sehr negativ für die Glaubwürdigkeit einer weiten Alien-Spielwelt, zu realisieren, dass man sich in einer 3D-Box befindet. Als Resultat entstand eine, persönlich als „Spielzeug-Grössenverhältnis" empfundene, nicht mit der Spielwelt kohärente Raumwahrnehmung.

Schlüsse:

- Immersion kann durch Kohärenz der Spielwelt erzeugt werden
- Wird die Kohärenz (visuell, auditiv, etc.) gestört, hat dies negative Auswirkungen auf die Immersion des Spielers
- Bugs[89] brechen die Immersion

Fast alle Testspieler erkannten zumindest eine Gemeinsamkeit von Spielelementen. Dieser Fakt macht alleine noch keine visuelle Einheit aus der Vielzahl der Spielinhalte. Jedoch fielen bei der umgekehrt gestellten Frage „Was in der Welt passt visuell nicht hinein?" kaum Landschaftselemente. Dies wird sehr positiv bewertet, da ein „Nicht-Auffallen" von Störfaktoren höchstwahrscheinlich ein Indiz für eine kohärente Bildsprache ist. Als häufigstes Element, welches nicht in die Welt passt, wurde der Astronaut genannt. Dies war durchaus gewollt, da der Astronaut aus einer anderen Welt (von der Erde) stammt und deshalb nicht ins visuelle Schema des Alien-Planeten Kepler-69c fällt (Vergleiche Kapitel 10.2). Da der Fokus aber auf einer Gesamtkohärenz aller Elemente abzielt, wird diese Erkenntnis über den Hauptcharakter trotzdem in die zukünftig zu lösenden Problemstellungen miteingegliedert.

Was den Spielspass anbelangt sind die Antworten sehr differenziert. Dies liegt mit hoher Wahrscheinlichkeit an den jeweiligen Präferenzen der Spieler. Die Spielereinteilung nach Bartle (Kapitel 1.4) liefert dazu weiterführende Auseinandersetzungen. Generell wurden Freunde der Exploration erwartungsgemäss besser bedient als Spieler, welche das Ziel hatten alle Aliens zu scannen. Da *GenoTerra* ein Open-World-Game ist, soll es besonders diese Explorer ansprechen. Schliesslich führten Bugs bei der Steuerung (Kameraführung) oder ein fast ganz fehlendes Belohnungssystem nach einem erfolgreichen Scannprozess (Kapitel 6.6) zur Frustration beim Spielen. Bemängelt wurde die geringe Anzahl von vier Kreaturen und das Fehlen einer Story, da zum Zeitpunkt der Playtestings noch kein einführendes Intro in die Story von *GenoTerra* (siehe 6.5) einführt. Beide Aspekte sind einfach durch Aufwenden von Zeit lösbar.

[89] Bug = Programm-, bzw. Softwarefehler

Schlüsse:

- Unbemerkt akzeptierte Inhalte sind der Beweis für eine kohärente Spielwelt
- Belohnungssystem(e) müssen integriert werden, um Spieler motiviert zu halten

7.3 Ausblick – Was wäre noch möglich mit diesem Prototyp

GenoTerra bedarf zum Zeitpunkt der Fertigstellung der Thesis diverser Verbesserungen (vergleiche 7.2). Die Ergebnisse des Playtestings, aber auch die persönliche Einschätzung sprechen dem Game-Prototyp durchaus Potential zu. Folgende Verbesserungen sind ohne weiteres bereits möglich:

- Story evaluieren und in *GenoTerra* einbinden
- Soundeffekte zu jedem Element hinzufügen (Schritte, Scanner, Kreaturen, etc.)
- Mehr Inhalte (Vegetation, Kreaturen, etc.) produzieren, um die Spielwelt interessanter und lebendiger zu machen
- Mechanik um verschiedene Features erweitern (Bspw. Leben von Astronaut, das bei Angriff eines Wesens abnimmt)

Für das Genre der Open-World-Games ist die steigende Rechenleistung der Computer eine sehr positive Entwicklung. Allgemein ist zu erwähnen, dass bei heutigem Stand der Computertechnik (die für Privatpersonen erschwinglich ist) nach wie vor Umwege bei der Programmierung genutzt werden müssen, um eine Berechnung der riesigen Spielwelten zu ermöglichen. *GenoTerra* löst dies durch das Deaktivieren von Terrainteilen, welche weit vom Spieler entfernt sind (siehe Kapitel 6.1). Solche Massnahmen könnten in Zukunft dank steigender Rechenpower nicht mehr so dringlich sein. Für *GenoTerra* würde dies konkret bedeuten, dass die Clipping-Distanz (vergleiche Kapitel 7.2) erweitert und Lags verhindert werden könnten. Auch die Anzahl der Aliens und anderer nicht-statischer Elemente kann einfach massiv erhöht werden, was der Spielwelt eine höhere Lebendigkeit verleihen würde. Zum Stichwort der prozeduralen Generierung in *GenoTerra* wäre eine grössere Vielfalt an Landschaftsformationen und Abstufungen mit mehr Ökosystemen möglich und wünschenswert.

Würde man *GenoTerra* umstrukturieren, ist aus persönlicher Sicht sogar eine Anwendung ausserhalb des Game-Kontextes möglich. Diese Aussage gründet auf der Vielfalt an Nutzungsmöglichkeiten der theoretisch unendlichen, virtuellen und begehbaren Welt von *GenoTerra* (siehe Kapitel 6.1). Eine Auswahl denkbarer Anwendungsbereiche mit Beispielen sind:

- Design (Wie müssen Dinge im 3D-Raum gestaltet sein um ...)
- Wahrnehmungspsychologie (Wie nehmen Menschen virtuelle Umwelten wahr?)
- Therapien (Höhen-, Raum-, und Platzangsttherapie)
- Schulvermittlung (Virtuelle Klassenfahrten in vergangene Erdzeitalter)
- Informatik (Wie gross kann die virtuelle Welt tatsächlich werden und wie verbessert man diesen Wert?)

8. Fazit – Persönliche Erkenntnisse zu prozeduralem Design ausgehend vom Experiment „GenoTerra“

In diesem Kapitel wird das Augenmerk auf die Konsequenzen dieser Thesis für persönliche Gestaltungsprozesse im Bereich der Spieleentwicklung gelegt. Ausserdem bildet es eine Auswahl verallgemeinerbarer Hinweise zur Nutzung von prozeduraler Generierung als Design-Ausgangslage für Spielinhalte.

8.1 Veränderungen im Designprozess durch prozedurale Generierung

Im Vergleich zu privaten, in der Vergangenheit realisierten Projekten im Bereich Game Design, wird die Nutzung von prozeduraler Generierung sehr erfolgreich als Hilfsmittel zur kohärenten Bildsprachenerzeugung gewertet. Die Gesamtästhetik erschloss sich im Gestaltungsprozess schlagartig, als die prozedural generierte Formensprache für den Grossteil der Inhalte genutzt wurde. Der Unterschied wird im Vergleich sichtbar.

Bild 90: GenoTerra Stand 23.09.2017. Die Elemente haben keine gemeinsame Designbasis

Bild 91: GenoTerra Stand 05.06.2018. Selber Ort in der Spielwelt. Die Elemente sind alle mit Ausgangsdesigns des Shape Generators erstellt

Verallgemeinernd kann gemutmasst werden, dass prozedural generierte Inhalte nicht automatisch zu kohärenteren Inhalten führen. Jedoch können sie dem Designer helfen eine Bildsprache zu entwickeln und diese erfolgreich als Gesamtästhetik im Game zu etablieren. *GenoTerra* dient dazu als Anwendungsbeispiel.

Damit ist die Hypothese **„Generatives Design ist eine neue Möglichkeit kohärente Bildsprachen in Open-World-Games zu erzeugen."** (Kapitel 1.8) im Rahmen der Untersuchungsmöglichkeiten dieser Arbeit **bestätigt**.

8.2 Vorteile von prozeduraler Generierung der Ausgangslage von Spielinhalten

Die Vorteile einer Nutzung von prozeduraler Generierung im frühen Designprozess der Spielinhalte erschliessen sich verallgemeinert bereits in Kapitel 1.7. Konkret aus der persönlichen Erfahrung mit dem Prozess in *GenoTerra* schliessend, kommen folgende Aspekte ergänzend hinzu:

- Designrichtlinien sind bereits in der prozeduralen Ausgangslage implementiert (d.h. im Fall von *GenoTerra* gab es bspw. eine sehr „fetzenartige" Formensprache)
- Eine solche implementierte Ästhetik kann, dank der Vorgabe der Shapes, einfacher im gesamten Projekt durchgezogen werden
- Generell fallen Designentscheidungen schneller und ermöglichen es, die Zeit für die manuelle Erstellung von ersten Konzepten (Ausgangslagen) in andere Bereiche (Ausarbeiten der Designs) zu transferieren

8.3 Nachteile von prozeduraler Generierung der Ausgangslage von Spielinhalten

Folgende Punkte wurden im Designprozess des Prototyps (im Kontext von prozeduraler Generierung) persönlich als negativ empfunden:

- Obwohl die Ausgangs-Shapes alle prozedural erstellt sind, waren aus der Perspektive eines menschlichen Designers trotzdem nicht alle verwendbar. Es entstand eine Selektion auf völlig freier Basis (Bauchgefühl)
- Ist eine Ästhetik erstmal etabliert, so lässt ein erstrebtes Höchstmass an visueller Kohärenz kaum Spielraum für andere Formensprachen zu. Dies ist nicht partout negativ (eher erwünscht), vereinfacht die Aufgabe eines Designers aber nicht
- Um mit prozeduraler Generierung Designs (bzw. deren Basis) zu erstellen, muss zuerst ein Programm/eine Programmierung erarbeitet werden, welche diese Designs erstellt. Das Schreiben eines Shape Generators (vergleiche Kapitel 5) kann im Extremfall mehr Zeit beanspruchen als das manuelle Erstellen von Inhalten. Hier würde sich prozedurale Generierung aber laut Kapitel 1.7 sowieso nicht lohnen.
- Ein solches Programm müsste womöglich auch an die Anforderungen neuer Projekte angepasst werden. Oft ist man dabei an den Programmierer im Team gebunden. Da *GenoTerra* ein Solo-Projekt war stellte dies kein Problem dar.

9. Persönliches Schlusswort

Zum Schluss bleibt zu sagen, dass das Programmieren und Designen mit dem Ziel einer möglichst hohen Bildsprachenkohärenz ein herausforderndes Unterfangen war, in welchem persönliche Interessen, Stärken und die Neugierde nach neuen Wegen des Designens von Open-World-Games zusammenkamen. Ich kann von dieser Erfahrung für meine weitere Arbeit im Bereich des Game Designs enorm profitieren und möchte den Prototypen *GenoTerra* in eine noch unbestimmte Richtung weiterentwickeln, da ich vom Potential solcher Projekte überzeugt bin.

10. Dokumentation:

10.1 Development Blog

Während meiner Arbeit an der Thesis habe ich einen Development Blog geführt, der mir zur Dokumentation des Prozesses diente. Im Blog werden die Entwicklungsschritte ebenso ersichtlich, wie die Hürden und Herausforderungen, die man in der Aneignung neuer Techniken und Methoden bewältigen muss.

Es ist sinnvoll den Development Blog online unter www.mnenad.com/devBlog aufzurufen um alle Inhalte (Videos, GIFs) abspielen zu können. Diese gedruckte Form dient der Vollständigkeit der Dokumentation.

illustration, art, game design

INTRODUCTION · 23. Januar 2018

DevBlog: Procedural World Generation in Unity?

Introduction

For the sake of documentation of my maser thesis in integrative design and also for sharing my achievements with a brother and interested audience I decided to create this development blog. The main topic so far, but this might change during the process of elaborating necessary fields for realising my intentions, is all about creating procedurally generated landscapes and do research about the meaning of the individual assets for the players. Mentioning "players" means, yes, I will make a game out of it. Or at least a working prototype that represents the thesis' final results in an applied case.
The engine I use is Unity (Link to Unity). I use this particular engine because I made other projects in it before and have enough knowledge to work fluently within its possibilities. Also, it uses the scripting language C# which I am familiar with (at a semi-advanced level I guess) since I started developing and designing virtual worlds.

I had in mind to swap to Unreal Engine for a while to not wrestle with Unity's build-in terrain engine later on (read about the problems here), but the short amount of time for the thesis and the motivation of getting to learn more about the software's possibilities outside the stiff build-in terrain engine made me stick with it. Obviously, I'm learning and therefor I use tons of references I will try to always mention the source correctly.

I will post about problems I encountered, solutions I found, designs I create and people I met (read what David O'Reilly "Everything" and Kate Edwards "Halo" answered to a question I asked them at Ludicious Game Festival in Zürich soon.)
I hope this will help others to get basic knowledge about procedural world generation in Unity and please let me know in the comment section what you think I could have done better/changed/whatever. Bring up your own ideas!

I will post updates to the project whenever something happened that should be shared with the world. Stay tuned!

Software I will use

As I mentioned Unity will be the engine. I use cinema4D to make my 3D-models, texture them and export them to Unity for further work. Of course Photoshop is essential for any illustrative or graphical part. I also use a Wacom tablet (Mobile Studio Pro) to draw and paint but as I also work inside Photoshop I'll keep this list short and don't go deeper into details.

And by the way I work most of the time on my Macbook Pro, so maybe no compute shaders (argh!)

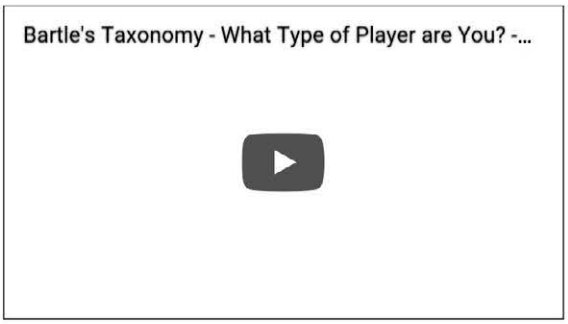
Who am I ?

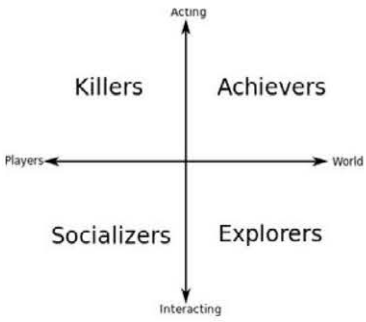
Read about it in the "about me" section. To cut a long story short: I'm a 24 y/o illustrator with a bachelor in art education. I loved games since childhood and began to develop my own digital experiments (check them out here) two years ago. I did a guest semester at game design department of the Academy of Arts in Zürich (Switzerland) and currently I'm finishing my master of arts in integrative design at the Academy of Arts Basel (Switzerland) with a strong focus on developing in Unity. I'm not a native English speaker so I already apologize for mistakes I make.

What is my motivation ?

My motivation is built up from my own experiences with explorative, digital game environments. According to Richard Bartle's classification of game types I'd say I'm more an explorer than any other of the types, even if one's never just one of them.

I love to hunt for creative designs and lovely crafted details, so as I enjoy exploring huge worlds, interact with them and get inspired by the developer's and designer's stunning work.
Have a look at the graphic I supplied or the great video explanation team of Extra Credit if you are not familiar with Bartle's taxonomy.





But now, hands on the theory!

What does "procedural generation" mean ?

I'd like to quote this definition form a paper I've read a month ago about procedural content generation. The authors of the first chapter, Julian Togelius, Noor Shaker and Mark J. Nelson, give an awesome introduction into a variety of techniques to generate assets and mechanics using non-human design agents. Download the full pdf here. it's absolutely worth having a read through.

The definition they give about the term procedural generation is as follows:

> The definition we will use is that "*procedural content generation" is the algorithmic creation of game content with limited or indirect user input. In other words, PCG refers to computer software that can create game content on its own, or together with one or many human players or designers.*
>
> (Togelius J./ Kastbjerg E./ Schedl D./ Yannakakis G.N)

The theoretical background

After I decided to make a project that includes landscape typology and picks out the player's relation to it as a central theme, I was searching for literature that could build the base, or at least give some theoretical input for the thesis. I found some mentionable sources I'd recommend to everyone who tackles the field of geography in her/his daily life.

- Tuan, Yi-Fu: Topophilia. A Study of Environmental Perception, Attitudes, and Values. New York: Columbia University Press 1974

Yi-Fu Tuan is a Chinese-American geographer that wrote about landscape typology and the human perception of it it through different cultures and time periods of homo sapiens' history.
Another book is a must-read for every architect (I've heard saying). A philosophical discussion about the abstract term "space" written by the french philosopher Gaston Bachelard.

- Bachelard, Gaston: Poetik des Raumes. Frankfurt am Main: Fischer Verlag 2017 (german version
- Bachelard, Gaston: The Poetics of Space. (english version)

I'm not linking this to any amazon offer. I suggest you support your local bookshop or check out the library in town :)
Both books gave me a base knowledge to think about questions one might ask in a theoretical discourse about fictional, digital environment and the interaction with the player or even perception of humans towards environments in general. Worth reading for sure!

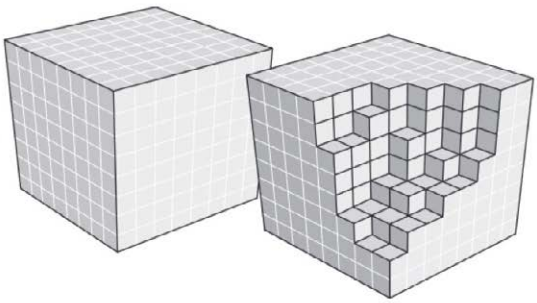
DEVELOPMENT · 24. Januar 2018

Start building the Landscape Generator

What kind of terrain do I want to create ?

I started the practical part by thinking of which technique would fit best in the end to allow the player to explore an endless generated world. I checked out some voxel based stuff and I knew from projects before (The PrättiApp) that it is fairly easy to generate a terrain by applying a heightmap on it. I decided against a voxel based system as I thought it would never be the case that the player manipulates the world's terrain directly like in Minecraft of No Man's Sky. But this means NO CAVES for the terrain and less interaction with it...

Voxels



In the examples above you can see how a player in minecraft is digging into the ground. This is achieved by destroying the cubes. Voxel terrains exist out of such individual pieces (in this case it is a cube, but it can also just be a vertex of a mesh like in the coming examples) that can be destroyed, replaced, or manipulated. Games like No Man's Sky use voxels to make mechanics like mining possible.

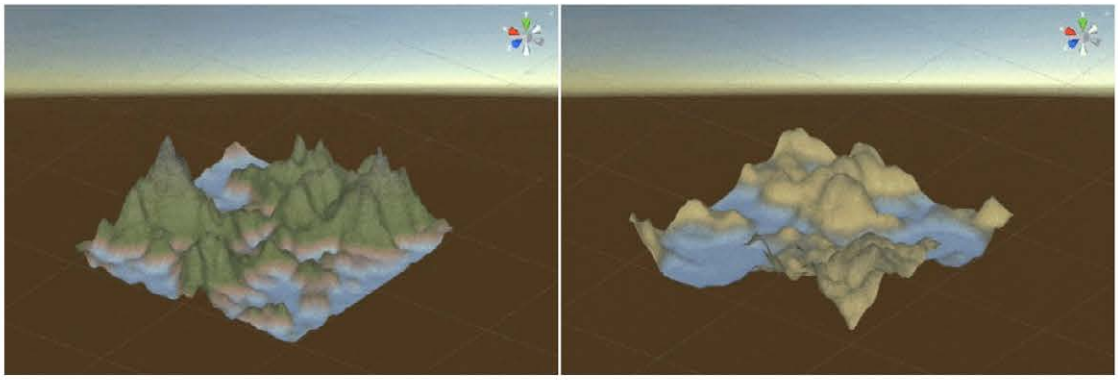
These examples show how smooth a voxel terrain can look in the end. For my understanding the shown examples look already a bit outdated, but from an developer's point of view I think it makes sense to show the base concept behind it.

The source for simple endless terrain

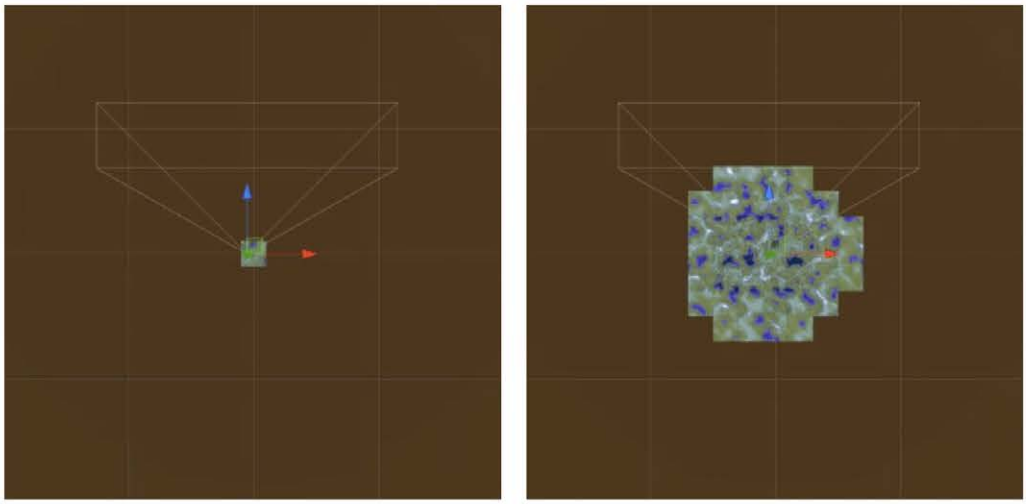
After plenty of research about endless terrains in Unity I found a promising tutorial series on YouTube about exactly what I wanted to achieve. The series is made by Sebastian Lague and whatever comes next would not be possible without his engagement in teaching and suppling up-to-date Unity tutorials. His explanations and code documentations are very clear and easy to understand. His code makes up the starting base for my terrain generator.

What Sebastian Lague teaches you:

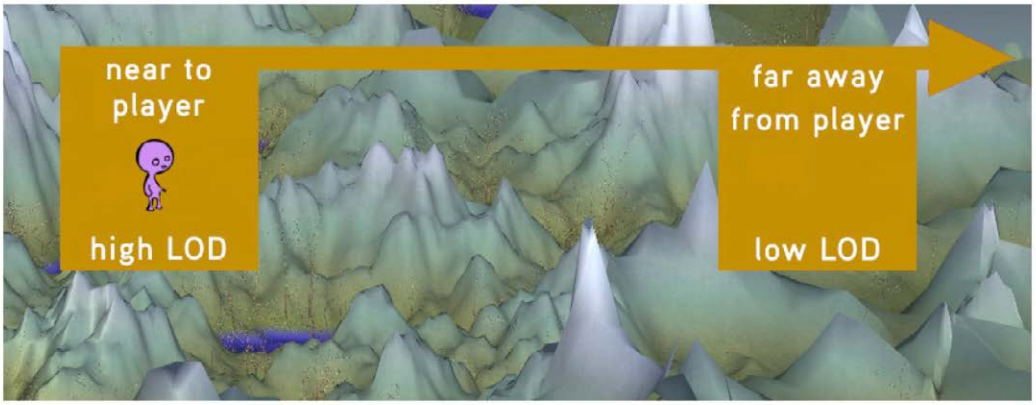
- creating editable terrain (you can modify the height, structure, details of the structure, textures, colours, colours according to height)
- creating shaders for editable terrain textures and color overlays



- generating seamless new terrain chunks when the player (the green moving centre point) walks around

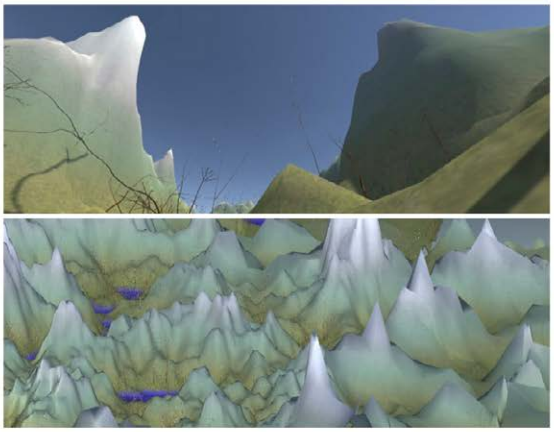


- implementing LODs (Level of detail). This is meant to reduce the details on terrain chunks that are far away and barely visible for the player to reduce computing resources needed

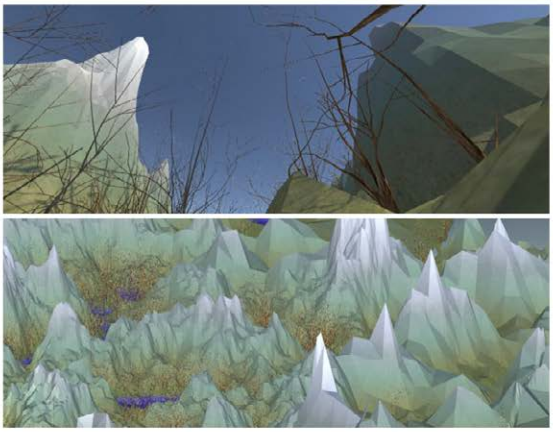


- create either islands or not falling-off terrains optional in a flatshaded style (ignore the trees, they are not part of it yet)

not flatshaded

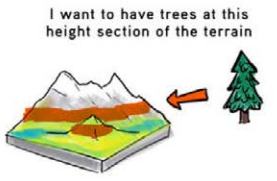


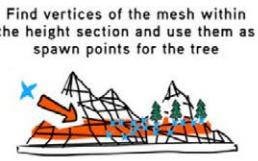
flatshaded

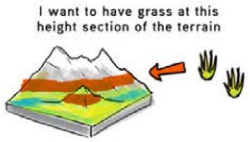


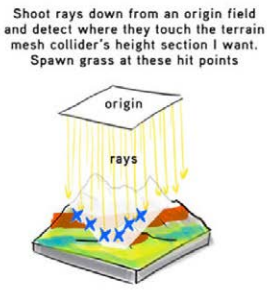
What Sebastian Lague won't cover (so far) – but I will:
Object/asset placement on the terrain...

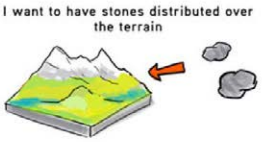
- ...by spawning on vertices depending on height of the individual vertex
- ...by spawning on raycast hit-points. Here I will use another tutorial by Sam Wronski, better known as World Of Zero on YouTube. I will cover it in a later post
- ...by placing the objects using a physics simulation (also developed by Sebastian Lague in another non related project but I extended the code to fit the needs of this particularuse case)

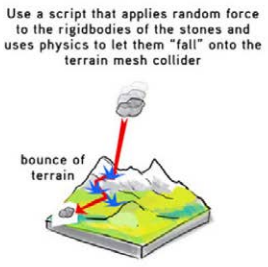


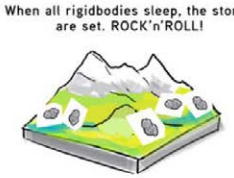












The essential tutorial by Sebastian Lague

I post the first video of the series right here, so you can find his YouTube-channel and get all the knowledge needed to continue later on:

And here you can find the whole terrain generator code on his GitHub (MIT license). I personally followed the tutorial step by step to understand what was happening in the various parts of scripts. This was, and still is essential to extend further functionality. Because his work was so much worth as a base to start with, I'll also give you a link to his Patreon page. If in any chance you can support him, do it:

whole Project on GitHub

Sebastian's Patreon page

DEVELOPMENT · 25. Januar 2018

Object Placement on Terrain Mesh

Before we start...

This chapter is all about how I solved it (so far) to be able to place all kinds of assets like 3D-meshes or self-growing fractal seeds on the terrain. I have to admit that this gave me a lot of headaches especially when it came to performance issues. Keep in mind that we create new chunks every time the player comes near the border of the chunk she/he is currently walking on. In fact, it obviously causes the more freeze the more objects one's trying to place at runtime and needs some optimization.

The biggest issue is the following:
Computations run by default on the so-called MAIN THREAD. It is simply a queue of executions that are calculated one after another. E.g.

```
// custom biom 1
if(pos.y >= 2 && pos.y <= 2.5f) // choose height section
{
    if (offsetMultiplier.x == 0 && offsetMultiplier.y != 0) {
        GameObject tree0Y = Object.Instantiate (treePrefab, new Vector3 (pos.x, pos.y, pos.z + assetOffSet * offsetMultiplier.y), Quaternion.identity);
        tree0Y.name = "Tree_at_X=0";
        tree0Y.transform.parent = treeParent.transform; // Placing + Parenting
    } else if (offsetMultiplier.x != 0 && offsetMultiplier.y == 0) {
        GameObject treeX0 = Object.Instantiate (treePrefab, new Vector3 (pos.x + assetOffSet * offsetMultiplier.x, pos.y, pos.z), Quaternion.identity);
        treeX0.name = "Tree_at_Y=0";
        treeX0.transform.parent = treeParent.transform; // Placing + Parenting
    } else if (offsetMultiplier.x == 0 && offsetMultiplier.y == 0) {
        GameObject tree00 = Object.Instantiate (treePrefab, new Vector3 (pos.x, pos.y, pos.z), Quaternion.identity);
        tree00.name = "Tree_at_XY=0";
        tree00.transform.parent = treeParent.transform; // Placing + Parenting
    }
}
```

The code calculates Instantiations (spawning assets with the given parameters), which is quite heavy for the system to calculate. Let's assume we do it for each vertex on a mesh with let's say 10'000 vertices. Guess what... if you don't have a very good machine it will struggle. This struggle is visible for the player as a freezing screen. Why?
Because the computer won't render the next frame until all the needed calculations are done!

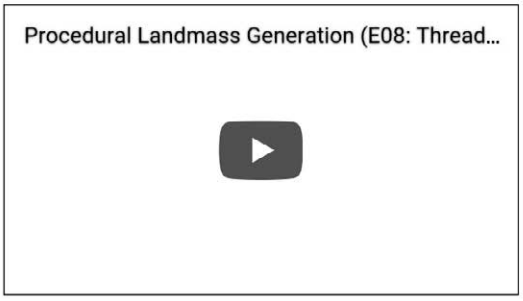
It's like you are watching a movie with friends at your place. At the most thrilling scene you get thirsty and go to kitchen to get an ice-cold beer. You pause the video and continue playing it when you are done with getting it. The time in-between your friends just see the paused image. It was frozen (kind of). It ruined the tension and part of the experience for your friends. At least bring some drinks for your friends as well next time...

And the same counts for a video game. Freezing is simply annoying and kills the flow.

Multithreading

What Sebastian Lague did to work-around it when generating new terrain chunks on runtime is to use multithreading. In the following video he explains it at 2:42min.

It goes way too far into detail to explain threading at this point and I am not sure if I get every part of it good enough myself.



Basically, creating a new thread means, you create a new internal coordination branch that can process calculations. This makes it possible to run two processes at the same time. The main thread has not to wait until all the objects are instantiated and can already render the next frame while the placement process runs on the separate, second thread. Here's a definition in other words from tutorialspoint.com:

A thread is defined as the execution path of a program. Each thread defines a unique flow of control. If your application involves complicated and time consuming operations, then it is often helpful to set different execution paths or threads, with each thread performing a particular job.

For more deeper information about multithreading have a look at tutorialspoint-link from above. I also found this forum post where a guy came up with a really nice tool. The ThreadingHelper. I will leave this downloadable zip-version here in case the link will break in the future.

UnityThreadingHelper
$UnityThreading.zip
Komprimiertes Archiv im ZIP Format [23.8 KB]

DOWNLOAD

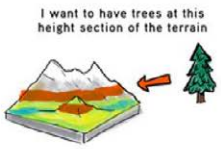
It is important to keep in mind that some processes like Object.Instantiate() or all things that are related to gameobjects like Gameobject.CreatePrimitiv so as Transform, aswell as void Start(), Update() and many others CANNOT BE CALLED OUTSIDE THE MAIN THREAD. This is also mentioned in episode 8 of Lague's tutorial series. You have to use abstract values for the parameters you want to change and apply the changes for example in the Update() function of your main thread.

Coroutines

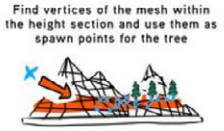
Another way of minimising lags is to use coroutines. Coroutines do not run on a separate thread, but they can give the main thread "time to breath". Let's say I call the method in the example below 50 times. Like I said, it would freeze cause the main-thread would calculate, and continue to render the next frame when it's done. By yielding back "WaitForFixedUpdate()" it will stop at that point of the code, let the main-thread working on the rendering of the next frame, and then get back as fixedUpdate is reached to finish it's task. This will not prevent freezing completely, but it can help to reduce it in my case!

```
IEnumerator RunCouroutineForPlacement()
{
    AutoGenerateComponents ();
    simulatedBodies = this.GetComponentsInChildren<Rigidbody> ().Select(rb => new SimulatedBody(rb, rb.transform.IsChildOf(transform))).ToArray();

    foreach (SimulatedBody b in simulatedBodies)
    {
        if (b.isChild && b.rigidbody != null)
        {
            float randomForceAmount = Random.Range (forceMinMax.x, forceMinMax.y);
            float forceAngle = ((randomizeForceAngle) ? Random.Range (0, 359f) : forceAngleinDegrees) * Mathf.Deg2Rad;
            Vector3 forceDirection = new Vector3 (Mathf.Sin (forceAngle), 0, Mathf.Cos (forceAngle));
            b.rigidbody.AddForce (forceDirection * randomForceAmount, ForceMode.Impulse);

        }
    }

    Physics.autoSimulation = false;

    for (int i = 0; i < maxIterations; i++)
    {
        yield return new WaitForFixedUpdate ();
        Physics.Simulate (Time.fixedDeltaTime);
        if (simulatedBodies.All (body => body.rigidbody.IsSleeping () || !body.isChild)) { //LAMBDA EXPRESSION

            break;
        }
    }

    Physics.autoSimulation = true;

    foreach (SimulatedBody body in simulatedBodies)
    {
        if (!body.isChild)
        {
            body.Reset ();
        }
    }

    RemoveAutoGeneratedComponents ();
}
```

Asset placement - depending on mesh height vertices



I showed you this picture last time, remember? Now this is what we will be implementing now. The key is to find all vertices of the terrain chunk's mesh and evaluate their height. If we have found each single vertex at the specific height we then need to instantiate the asset at the vertex's position vector.



So, first off let's find the part of the script where the chunks are being generated. This is the place for getting the vertex and of course its position vector because here the LOD (Level of detail) mesh is being applied, so we can grab the information we need right at it's source.

It is found at:

TerrainChunk (script) public void UpdateTerrainChunk()

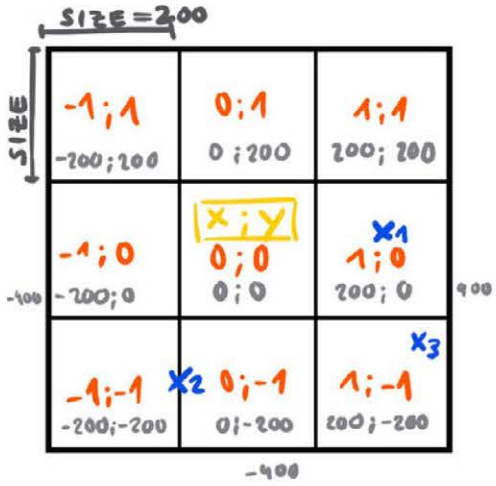
```
public void UpdateTerrainChunk()
{

// Gras Updaten
AssetPlacement.StartGrasUpdate ();

    if(heightMapReceived)
    {
        // bounds.SqrDistance() gibt die kürzeste Distanze im Quadrat zwischen zwei Punkten aus
        // Mathf.Sqr --> Zieht Quadratwurzel --> Hier also Resultat = Distanz
        float viewerDstFromNearestEdge = Mathf.Sqrt(bounds.SqrDistance(viewerPosition));
        bool visible = viewerDstFromNearestEdge <= maxViewDst; // Bool ob man noch IN oder schon AUSSERHALB der Maximalen Distanz ist

        bool wasvisible = IsVisible (); // wird true wenn der Chunk im letzten Update sichtbar war
        //viewerDstFromNearestEdge und Threshold von LOD vergleichen um zu erfahren, welcher Chunk mit welchem LOD angezeigt werden soll
        if(visible)
        {

            int lodIndex = 0; // Index des lod

            for (int i = 0; i < detailLevels.Length -1; i++) // detailLevel.Length -1 weil bei detailLevels.Lenght wäre der Chunk inaktiv (viewerDstFromNearestEdge <=  maxViewDst --> visible = false ))
            {
                if (viewerDstFromNearestEdge > detailLevels [i].visibleDstThreshold) // Wenn man(viewer) weiter weg ist, als lod-Threshold wird Auflösung kleiner(lod = höher)
                {
                    lodIndex = i + 1; // Erhöhe Vereinfachung wenn weiter weg
                } else
                {
                    break; // Wenn man sich im korrekten LOD-threshold befindet passiert nichts
                }
            }
            // Wenn der LOD der Chunk sich verändert hat im Vergleich zum letzten lod(previousLOD) --> (man hat sich weit genug bewegt) soll er updaten
            if (lodIndex != previousLODIndex)
            {
                LODMesh lodMesh = lodMeshes [lodIndex];
            if (lodMesh.hasMesh) //Wenn das LodMesh-struct bereits ein Mesh hat
            {
                previousLODIndex = lodIndex; // Werte neu ausgleichen
                meshFilter.mesh = lodMesh.mesh; // Neues Mesh mit richtigen LOD generieren

                // Wenn man alle Eckpunkte (verteces) kennt, kann man die Assets an ihnen spawnen
                foreach (Vector3 vertex in meshFilter.mesh.vertices)
                {
                    // Platzieren der Bäume anhand von den Eckpunkten des Meshs, das hier generiert wird mit der Grösse der Map, also dem ganzen Offset (-> Verschiebung der Assets)
                    // und der halben Mapgrösse, also dem halben Offset (-> für Zentrum des Chunks --> 0-Punkt-Coordinaten zum Spawnen) und dem Baum-Objekt an sich
                    // coord wird als Multiplikator für den offset genutzt (0,1,2,3 - 0,-1,-2,-3 usw.)
                    AssetPlacement.SpawnAssetsOnChunkVerts(vertex, sampleCenter, meshSettings.meshWorldSize, coord, meshSettings.tree, meshFilter.transform);
                    AssetPlacement.SpawnAssetsTwoOnChunkVerts(vertex, sampleCenter, meshSettings.meshWorldSize, coord, meshSettings.treeTwo, meshFilter.transform);

                    // Höhe für Schilfgras (Reed)
                    if (vertex.y > 0 && vertex.y < 2 && !alreadyGeneratedObjectAtThisVertexCoords.Contains(new Vector2(vertex.x, vertex.y)))
                    {
                        AssetPlacement.SpawnReedOnChunkVerts (vertex, sampleCenter, meshSettings.meshWorldSize, coord, meshSettings.reed, meshFilter.transform);
                        alreadyGeneratedObjectAtThisVertexCoords.Add (new Vector2 (vertex.x, vertex.y));
                    }
                }

            }
```

We could already place the if-statement here to choose a certain height, like in the last of the three calls. No idea why I did not do it instantly for all of them, but I guess It might been my intention to have one external script, the so-called AssetPlacement (script) to check all height conditions for all type of assets I wanted to include in the near future.

As you can see AssetPlacement.SpawnAssetsOnChunkVerts() takes in a bunch of parameters. Those are:

- vertex = the Vector3 vertex position (x, y (height), z)
- sampleCenter = the Vector2 coordinates of the chunk
- meshSettings.meshWorldSize = size of the chunk
- coord = this is a offset-value achieved by using the basic matrix of an cellular automaton grid
- meshSettings.tree = reference of the prefab of the asset we want to spawn
- meshFilter.transform = reference of the asset's parent (so it's only active in the scene when the chunk is)

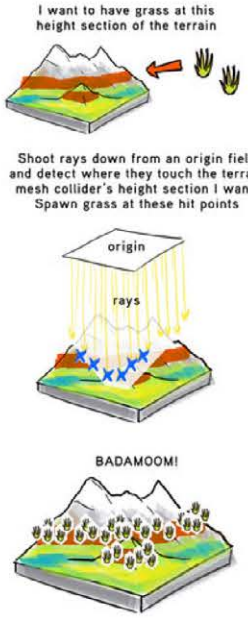
I used the meshSettings reference because TerrainChunk is not a Monobehavior and can't access public gameObjects in the inspector. But TerrainChunk contains the MeshSettings class reference and can easily be accessed. A bit of a work around I have to admit.



Assuming the size of a terrain chunk is 200 x 200 units, the assetOffset of its centre is always minus or plus the amount of sizes to the left/right, up-, and downwards. This assetOffset is our sampleCentre parameter and it is translated into a value that tells us how many terrain chunk units we're away from centre as well as in which direction. The so-called offsetMultiplier (the coord value) becomes important when we want to place objects on all terrain chunks and not only on one of them, because it means we can simply multiply the sampleOffset by the offsetMultiplier. The chunk in the middle is at 0;0. All others arrange according to the vector grid (x = width / y = height). So, it's 0 x 200 on the x-axis and 0 x 200 on the y-axis. If the vertex position was something like Vector3(12, 2.3, 56) the asset will be instantiated at this Vector3 coordinate. When we check the chunk at 1;-1 we can simply add Vector3(12, 2.3, 56) + offset(sampleCentre) x multiplier(coord). Easy!
The AssetPlacement (script) contains the actual spawning of the asset on the chosen height of the mesh vertices.

```
public static void SpawnAssetsOnChunkVerts(Vector3 pos, Vector2 centre, float assetOffSet, Vector2 offsetMultiplier, GameObject treePrefab, Transform treeParent)
{
    #region TREES 1 (zwischen y = 2 und 2,5)
    // custom biom 1
    if(pos.y >= 2 && pos.y <= 2.5f) // Werte für Biom-Höhengrenzen festlegen
    {
        if (offsetMultiplier.x == 0 && offsetMultiplier.y != 0) {
            GameObject treeOY = Object.Instantiate (treePrefab, new Vector3 (pos.x, pos.y, pos.z + assetOffSet * offsetMultiplier.y), Quaternion.identity);
            treeOY.name = "Tree_at_X=0";
            treeOY.transform.parent = treeParent.transform; // Placing + Parenting
        } else if (offsetMultiplier.x != 0 && offsetMultiplier.y == 0) {
            GameObject treeX0 = Object.Instantiate (treePrefab, new Vector3 (pos.x + assetOffSet * offsetMultiplier.x, pos.y, pos.z), Quaternion.identity);
            treeX0.name = "Tree_at_Y=0";
            treeX0.transform.parent = treeParent.transform; // Placing + Parenting
        } else if (offsetMultiplier.x == 0 && offsetMultiplier.y == 0) {
            GameObject tree00 = Object.Instantiate (treePrefab, new Vector3 (pos.x, pos.y, pos.z), Quaternion.identity);
            tree00.name = "Tree_at_XY=0";
            tree00.transform.parent = treeParent.transform; // Placing + Parenting
        }
        else if(offsetMultiplier.x != 0 || offsetMultiplier.y != 0)
        {
            GameObject treeXY = Object.Instantiate (treePrefab, new Vector3 (pos.x + assetOffSet * offsetMultiplier.x, pos.y, pos.z + assetOffSet * offsetMultiplier.y), Quaternion.identity);
            treeXY.name = "Tree_value";
            treeXY.transform.parent = treeParent.transform; // Placing + Parenting
        }

    }
    #endregion
}
```

grass placement - depending on raycasting hit points



Tutorial time again. I followed World of Zero's tutorial on how to achieve this and added some lines of code to make it more flexible for my particular case. His series goes further but I needed to quit after part 3. I have absolute no clue what caused the issue, but I was not able to sum up the way to long and unclear arranged geometry calculation in the shader within a single function. He Sam Wronski does it in part 4 at the very beginning. Yes, I checked case sensitivity like 100 times... I thought it might be a bug that doesn't support triStream.RestartStrip() ??

And unfortunately compute shaders are not supported on Apple machines... argh. Anyways... I found out I don't know to code shaders with CG (Unity's shader scripting language) and so until today I haven't found a way to get shadows and other lighting stuff working on this shader. I posted my questions on some relevant helping platforms. But no success so far...

Read about them on stackoverflow.com

The most important scrip besides the shader itself is the GrassRenderer. I post a pic with some comments in german right here. But I suggest you, as always, to follow the YouTube series to get what it is all about.

```
void GeneratePointCloud()
{
    Random.InitState (seed); // Random.InitState = Initialisierung von Random mit einem seed (gibt also eine immer gleiche, zufällige Zahl)

    //Liste(mit Grösse = grassNumber) von Positionen, Array aus Indexen der Positionen und Farben-Liste für jeweilige Position (für Grasvariation)
    List<Vector3> pointPositions = new List<Vector3>(grassNumber);
    int[] pointPosIndecies = new int[grassNumber];
    List<Color> grassColorAtPoint = new List<Color> (grassNumber);
    List<Vector3> normals = new List<Vector3> (grassNumber);

    for (int i = 0; i < grassNumber; i++)
    {
        Vector3 origin = transform.position; // Eigener Startpunkt (Höhe)
        origin.y = startRaycastHeight; // Zu Beginn ist diese Höhe = der Reycasthöhe
        origin.x += size.x * Random.Range (-0.5f, 0.5f); // Halbe länge in die eine Richtung = Rechteck. 0.5 kann vllt durch size.x/2 ersetzt werden
        origin.z += size.y * Random.Range (-0.5f, 0.5f); // ...und halbe länge in die andere Richtung. 0.5 kann vllt durch size.x/2 ersetzt werden

        Ray ray = new Ray (origin, Vector3.down); // Ray von Punkt "origin", gerade nach unten (Vector3.down)
        RaycastHit hit; // Treffpunkt des Rays auf Objekt

        if (Physics.Raycast (ray, out hit, 1000, layerMask)) // Wenn der Ray etwas trifft...
        {
                origin.y = hit.point.y; // ...wird dieser Treffpunkt zur Höhe
                origin.y += grassOffset;
            if (origin.y > upperGrasBorder || origin.y < lowerGrasBorder)
            {
                origin.y = -10;
            }
                origin.x -= this.transform.position.x;
                origin.z -= this.transform.position.z;

            pointPositions.Add (origin); // Position für mesh
            pointPosIndecies[i] = i; // Indexe der Positionen des Meshs
            grassColorAtPoint.Add(new Color(Random.Range(0.0f,1.0f), Random.Range(0.0f,1.0f), Random.Range(0.0f,1.0f),1)); // Random Farben für Mesh an jew. Positionen
            normals.Add(hit.normal); // Normale des getroffenen Punktes returnen

        }
    }

    // NEUES MESH MIT WERTEN (SET...) ERSTELLEN !!!
    pointCloudMesh = new Mesh ();
    pointCloudMesh.SetVertices (pointPositions);
    pointCloudMesh.SetIndices (pointPosIndecies, MeshTopology.Points, 0, false); // Art des Meshes (MeshTypologie muss festgelegt werden und Submesh = 0, wenn keines da)
    pointCloudMesh.SetColors (grassColorAtPoint);
    pointCloudMesh.SetNormals (normals);
    grasMeshFilter.mesh = pointCloudMesh;
}
```

The implementation for using this system on the terrain chunks that are generated is:

```
public void PlaceObjectsOnChunks()
{
    foreach (Transform t in this.transform)
    {
        // If the Transform-object has a collider and the TRansform is yet not in the List...
        if (t.GetComponent<MeshCollider> ().sharedMesh != null && !alreadyGeneratedObjectAtThisChunkTransform.Contains(t))
        {
            // ...execute this function
            AssetPlacement.SpawnGrassGeneratorAtChunkPosition (t.transform.position, grasGeneratorPrefab, t);
            AssetPlacement.SpawnGrassGeneratorAtChunkPosition (t.transform.position, weedGeneratorPrefab, t);
            AssetPlacement.SpawnPhysicsSimulatorsAtChunkPosition (t.transform.position, stoneSimulatorPrefab, t);
            AssetPlacement.SpawnPhysicsSimulatorsAtChunkPosition (t.transform.position, rockSimulatorPrefab, t);
            AssetPlacement.SpawnWaterAtChunkPosition (t.transform.position, waterPrefab, t, waterLevel);
            // same function as SpawnGrassGeneratorAtChunkPosition with other prefab

            alreadyGeneratedObjectAtThisChunkTransform.Add (t); // Add Transform to the List (to be checked in next iteration)
        }
    }
}
```

We only want the gras to be placed on chunks that have a collider attached to them. The reason is that raycasting works only on the collider and not the mesh itself! So therefor you can simply wait until all chunks are generated and then loop through them and check which do already have a collider attached. The code snipped above is then just used on these chunks. But what does it actually do?
It places the prefab, you learn to create in the tutorial series by World of Zero, over the chunks and raycast down on them to find the points for spawning grass on. Or... weed (lol). The same works (without the raycasting of course) also to place assets like water planes(Unity Environmental package) on the terrain. It's a bit easier as we don't need to find all the vertices to place the assets first.

Here you also see the alreadyGeneratedObjectAtThisChunkTransform.Add (t) **which is a way of checking if the chunk that is processed already has this objects placed! Otherwise we instantiate every time we update the chunks also the assets again. We check it by using a HashSet.**

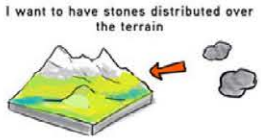
```
// Check if instance at this point already exists --> if YES, don't instantiate another.
HashSet<Transform> alreadyGeneratedObjectAtThisChunkTransform = new HashSet<Transform>();
```

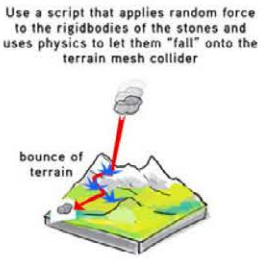
When we use our placement method on this chunk, its Transform component gets added to the HashSet. Next time we check if the HashSet contains the Transform values for this chunk. If it does: skip it.
This is also used in other contexts. Basically whenever a "have-I-already-done-that-on-this-chunk?" question appears.

asset placement - depending on physics

Last but not least: Physics.

By adding a Rigidbody and Collider component to assets like stones, while they are somewhere high over the terrain, we can let them fall down and make them act like “real” bodies on a terrain. There will tumble down,





collisions will be calculated and as soon as the assets stays still on the ground – the rigidbody sleeps – the rigidbody and maybe also the collider becomes unnecessary and can be deleted. Safes us resources. By the way I think it makes sense to use BoxColliders instead of some complex MeshColliders. We are anyway running on the limit with calculating physics on runtime!

The thing with "maxIterations" somehow did not turn out to work properly when working with Coroutines! I used them for the reason of lag reducing I talked about earlier in this post. So, what I did: I made a separate script that destroys the asset if it falls through the terrain! AND THAT HAPPENS! BECAUSE OF REASONS!
Ya, I think it's phenomenon caused by a collision detection issue and known among the Unity community.

To achieve the promised stuff follow this great one by Sebastian Lague:

[Unity 2017] Placing Objects with Physics

And this is my way of getting rid of the assets that fall through the terrain into eternity:

The `BrokenAssetDestroyer` (even if there's nothing broken on them)

```
public class BrokenAssetDestroyer : MonoBehaviour {

//      void Update()

    public void DestroyAssetInSeperateClass(GameObject asset)
    {
        asset.SetActive(false);
    }
}
```

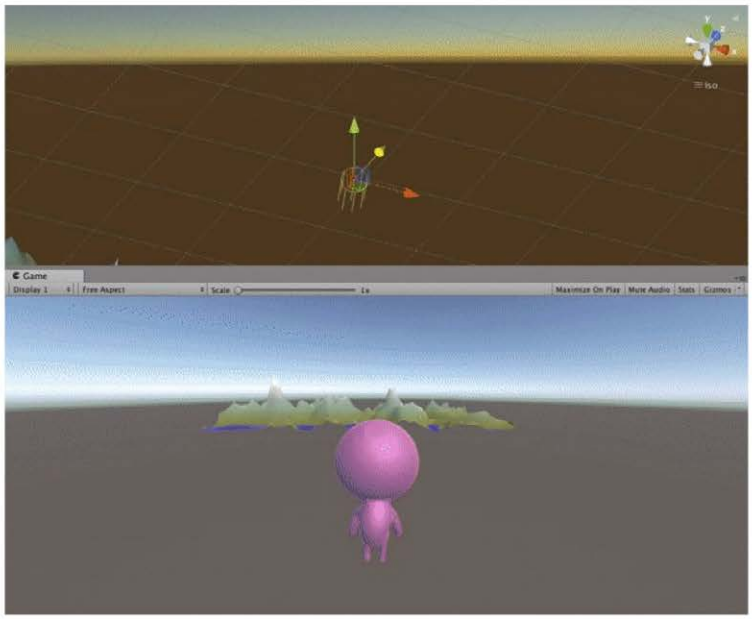
The `void Update()` in the placement script

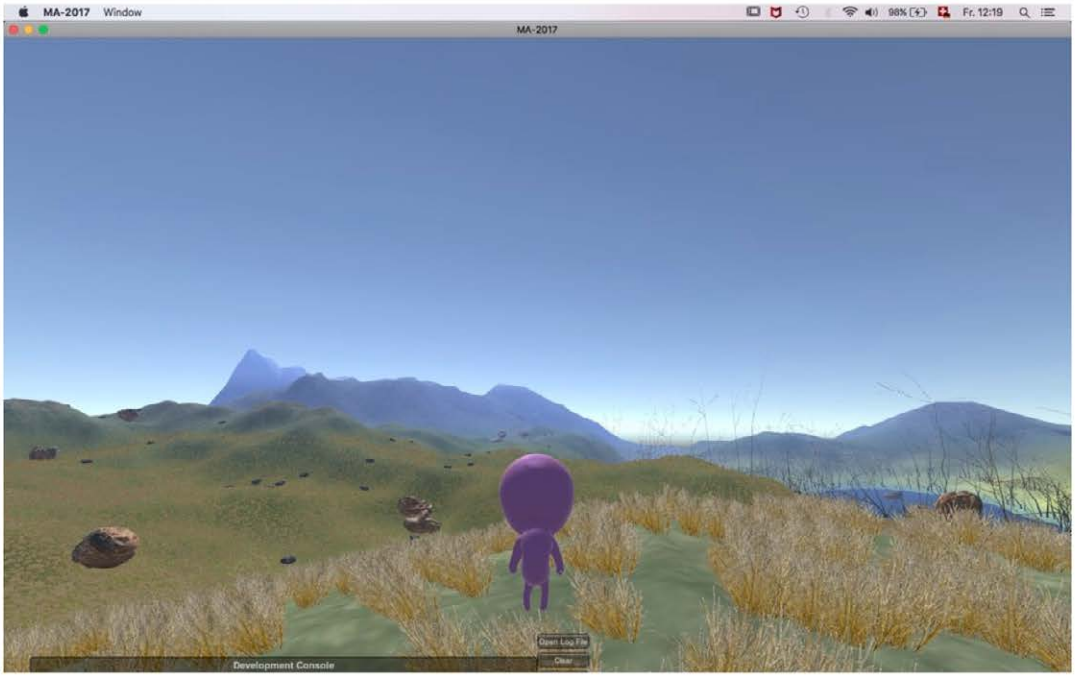
```
void Update()
{
    CheckHeightOfAssets ();
}

void CheckHeightOfAssets()
{
    foreach (Transform child in transform)
    {
        if (child.transform.position.y < -150)
        {
            brokenAssetDestroyer.DestroyAssetInSeperateClass (child.gameObject);
        }
    }
}
```

You already know how to instantiate this objects on the terrain chunks with that contain colliders already form the previous placement methods. But in addition you can for example spread the starting point for each asset a bit. This helps also to prevent that one asset falls on another if they don't have the same starting height! In my project this was the cause of weird flying stones! The upper stone fell onto the stone beneath it and as the rigidbody fell asleep, it got destroyed and the stone rested peacefully in heaven. Literally!

the example scene

That's an example of how the generation and placement of assets works hand in hand. Hope you could get the concept behind the methods and scripts I used. I'm looking forward to read some feedback!



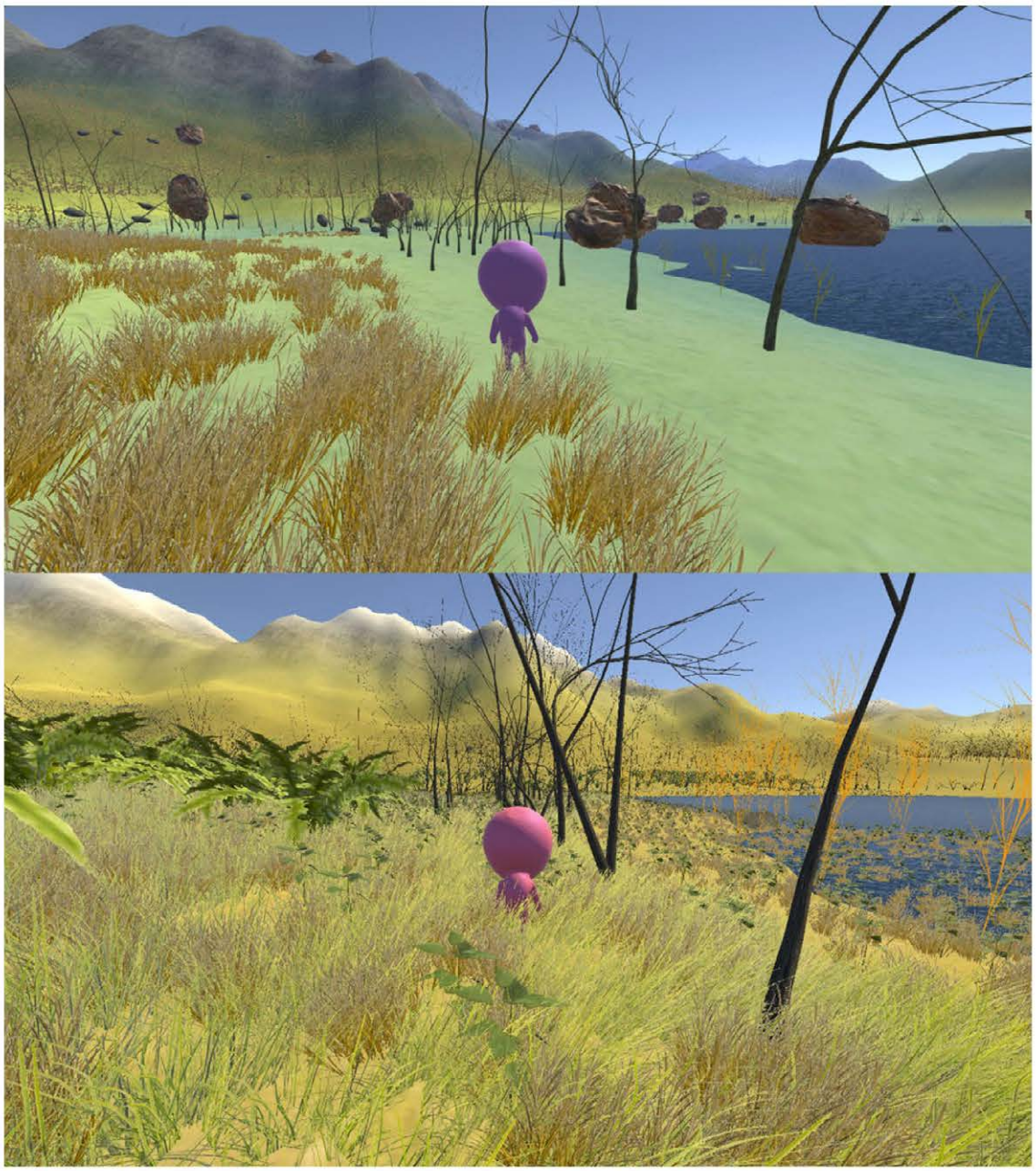


After tweaking around with the lacunarity and size values as well as adding fog through the built in Unity Lighting Settings --> you will find the Fog-tab, I made a fairly nice looking screenshot of the in game view. I know there's tons of improvements to work on, but for a first impression I'm kinda motivated to go on!

DEVELOPMENT · 26. Februar 2018

Optimize Vegetation Generation

Step by step to dense vegetation and acceptable FPS



Since the last post I've been working on the terrain's vegetation. Mainly I focused on generating grass that bends in the wind and some fern like plants, but what comes next is usable for all kind of meshes. The big problem was (and still is in some cases):
HOW CAN WE GENERATE DENSE VEGETATION AND PLACE IT FAST AND WITHOUT TO MUCH FPS DROP AFTER THE GENERATION PROCESS ?

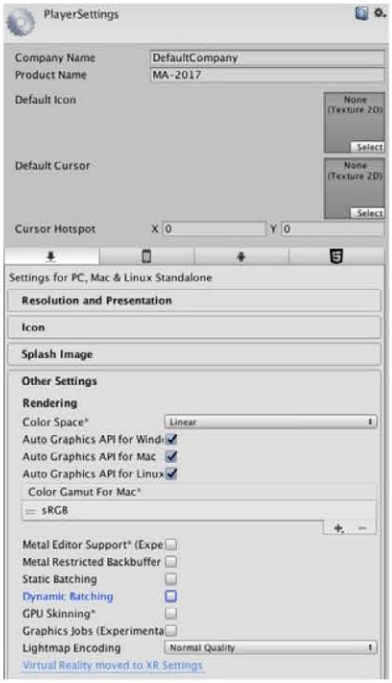
I have to admit it was frustrating to work with a machine that cannot handle much heavy calculations at once (my Macbook from 2011) from time to time, but on the other hand this forced me to get into optimizing the game very soon, what might become a good base for expanding the complexity in near future.

Draw call batching

First off: I learned a lot about Unity's build in batching system. Batching means to combine mesh objects that share the same material or that are marked as static in the Unity inspector.

The whole point here is that it reduces the amount of draw calls.

A draw call is the amount of materials on objects that are drawn each frame in the scene.
So assuming:



- 100 not batched objects all having the same material = 100 draw calls
- 100 batched objects with the same material = 1 draw call

Batching makes the more sense, the more objects you have on screen at the same time. So, if the goal of this chapter is dense looking vegetation on the landscape's hills – batching is definitely the way to go. Please note that the fact how well your computer handles draw calls and how much they affect the FPS (frames per second) is depending on your hardware. More details can be found under Unity's official documentation page for Draw Call Batching:

Draw calls are often resource-intensive, with the graphics API doing significant work for every draw call, causing performance overhead on the CPU side. This is mostly caused by the state changes done between the draw calls (such as switching to a different Material), which causes resource-intensive validation and translation steps in the graphics driver.

Unity uses two techniques to address this:

- **Dynamic batching:** for small enough Meshes, this transforms their vertices on the CPU, groups many similar vertices together, and draws them all in one go.
- **Static batching:** combines static (not moving) GameObjects into big Meshes, and renders them in a faster way.

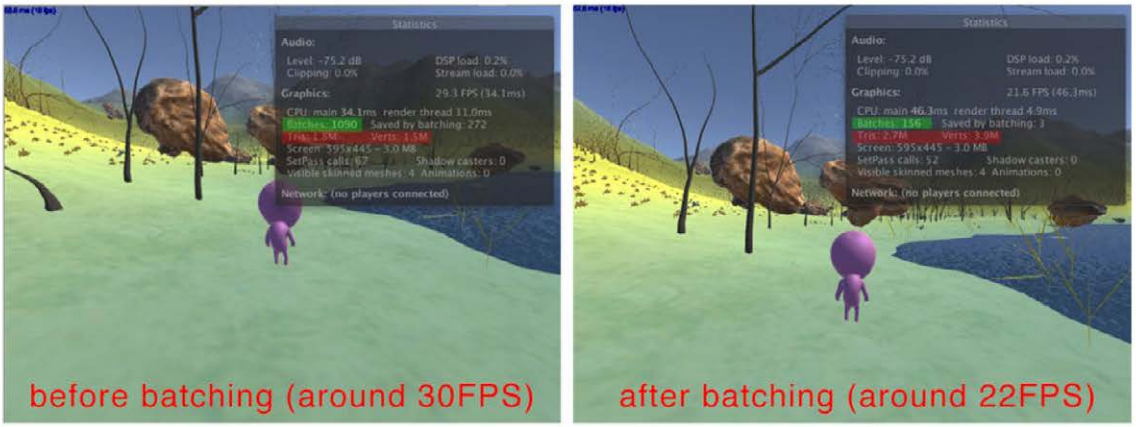
Side note:
Watch the clip in this link if you're not familiar or just interested in understanding the difference between CPU and GPU. Short: CPU has just a few big calculation units (cores) that work very well sequential but not parallel while GPU has thousands of small cores that run well parallel but not sequential.

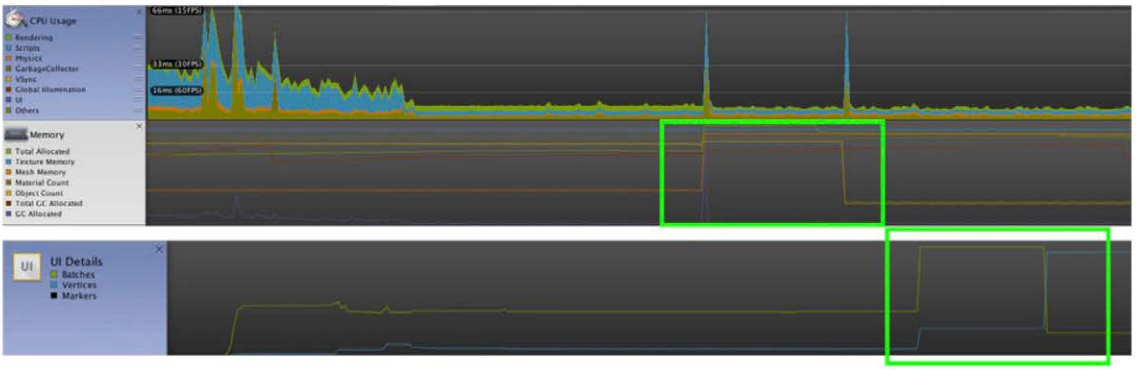
The core experience I made with Unity's batching system: it works. But there are way better solutions for not much money available. In my case I had terrible FPS with just some thousand mesh instances.

Batching on runtime

Another critical thought was that I want to batch the vegetation meshes and some other meshes on runtime while playing the game. Doing it before starting the actual game would mean to already generate all the content that needs to be batched at the beginning, which is no solution for a theoretically "endless procedurally

generated" terrain. Therefor I bought a product from Unity's Asset Store called Runtime Mesh Batcher (Which seems no longer to be available). It was easy to set up and on low cost. The creator Mr. Georges Dimitrov form Concordia University was a huge help, as I explained him by mail some trouble I got when using the asset on Unity version 2017.3.1f1. He updated the asset one day later which I consider great support. The main problem that remained and I discovered using the Profiler was that batching the meshes on runtime generates extra vertices and triangles. These are not physically there but get allocated in memory during the combining process - which is NEVER freed completely by Unity until the application is quit.





You can see the amount of vertices go up while the batches go rapidly down in the green framed areas. This was recorded during the batching process.
If you just need to batch once or twice during runtime even a high number of objects (in a demo scene he uses 10k mesh objects) the FPS will increase immensely with the code provided by Mr. Dimitrov, but if you do it over and over again I recommend MeshCombineStudio. It is slightly more expensive (it was on sale when I got it – Yehaw!) but you have the option to batch whenever you want without adding extra verts and tris which can be checked in the stats window in the editor. You can even delete triangles that are never visible by the player and delete vertices under certain layers. The only thing I couldn't really understand immediately and in my opinion, isn't marked very clear, is that all meshes one wants to batch need to be child objects of a defined parent. MeshCombineStudio will, unlike RuntimeMeshBatcher that does a better job in this point, only search for batchable objects in this one parent. But you can have multiple MeshCombineStudio instances as a workaround option.

Nevertheless, I also spent some money on the Advanced Foliage Shaders v.5. Simply to get rid of the annoying fact of not being able to handle the grass geometry shader I wrote about in the last post, due to my poor CG programming knowledge. This asset contains great foliage shaders with lot of options and finally some nice wind bending for the leaves and even touch bending. That's it. When batched, the animations from the Advanced Foliage Shader go a bit crazy and make the grass float around. I just toggled the "Baked Pivots" option in the shader to ON. This caused the grass to lay flat on the ground as long as the batching process is not completed, but this isn't visible from the distance anyways and eliminates the floating. The fern I made needed a bit more of fine tuning. It simply depends on the mesh and your individual set up.

Remodel 3D meshes (in Cinema4D)

Something that seems to be obvious but I cannot point out enough is: KEEP YOUR MESH VERTS LOW! I figured out that I was trying to spawn 60k meshes with an average of around 1000 vertices. Draw calls without batching were around 80 million. No wonder I had less than 0.6 FPS on my machine. So, I started to remodel some meshes in Cinema4D, my prefered 3D modelling software.

Some very basic tips:

- Rather be sure how much polygons you need before-hands and keep it as low as possible
- Use bump-, or normalmaps as well as displacement maps instead of modelling every detail
- The polygon reduction object from Cinema4D does not reduce the polygon count effectively
- Subdividing the mesh in Cinema4D's sculpting mode and baking a displacement map works great

For example, this is how a regular grass mesh would look when exported as fbx from Cinema4D into Unity. And it has just twelve vertices. The foliage shader has an alpha cutout value and uses a texture + culling set to off (it is visible from both sides of the planes).

Of course, the fern mesh has many more than the grass model, but just keep it as low as possible.

Just draw the meshes instanced - fast as hell but limited

Now that the batching works I decided to come back to a Script I got from World Of Zero's Youtube-tutorial that actually is an old version of the current grass generator script. It uses the Graphics.DrawMeshInstanced() function to draw a maximal amount of 1023 meshes (the max. number it can handle) at time with by GPU instancing.

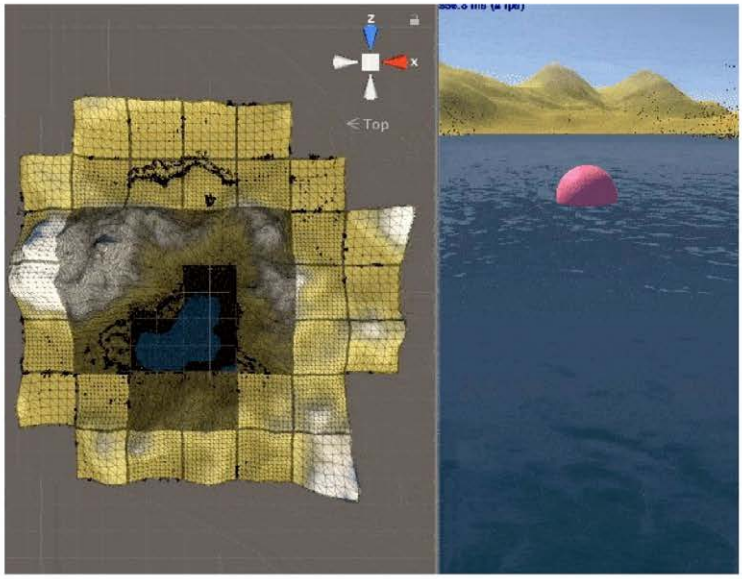
- GPU instancing: Use GPU Instancing to draw (or render) multiple copies of the same Mesh at once, using a small number of draw calls. It is useful for drawing objects such as buildings, trees and grass, or other things that appear repeatedly in a Scene. (read more)

The solution to its limitation was to have several game objects with each 1023 instances of the mesh and add them to one parent which acts as a spawnable prefab at each chunk position just like in the post about placement of objects I made. All these techniques combined already give an idea of living, dense nature.

```
using System.Collections.Generic;
using UnityEngine;

[ExecuteInEditMode] // Can be commented out when used on runtime
public class MeshGrasRenderer : MonoBehaviour {

    public Mesh grassMesh;
    public Material material;

    public int seed;
    public Vector2 size;

    [Range(0,1023)]
    public int grassnumber;

    public float startHeight = 1000;
    public float raycastMaxDist = 1000;

    public LayerMask layermask;

    void GenerateGrassMeshes()
    {
        Debug.Log ("Executing GrassGenerator");
        Random.InitState(seed);
        List<Matrix4x4> materices = new List<Matrix4x4> (grassnumber);

        for (int i = 0; i < grassnumber; i++)
        {
            Vector3 origin = transform.position;
            origin.y = startHeight;
            origin.x += size.x * Random.Range (-0.5f, 0.5f);
            origin.z += size.y * Random.Range (-0.5f, 0.5f);

            Ray ray = new Ray (origin, Vector3.down);
            RaycastHit hit;

            if(Physics.Raycast(ray, out hit, raycastMaxDist, layermask))
            {
                origin = hit.point;
                materices.Add (Matrix4x4.TRS (origin, Quaternion.identity, Vector3.one));

            }

        }
        Graphics.DrawMeshInstanced (grassMesh, 0, material, materices);
    }
    // Update is called once per frame
    void Update ()
    {
        GenerateGrassMeshes ();
    }
}
```

De/activate vegetation and meshes on chunks

To optimize the generation process it makes sense to first batch, and then deactivate assets that are to far away to be noticeable for the player. Therefor one can use a threshold to determine the distance between each chunk and the player and set the values when de-, or activation should happen through gameObject.SetActive (true/false).



Summing it up I can run the game with:
(2 x 60k grass instances by grass geometry shader) + (16 x 1023 mesh instances by graphics.drawmeshinstance function) + (2 x 60k gras mesh placement with raycasting) + (more than 5000 x more complex fern meshs with raycasting) + (2 x 10 stone instances through physical placement) + (I guess around trees 300 with vertices height check) = around 250k meshes PER CHUNK!

Which is quite nice. Especially because the FPS are on an acceptable level. Even when running on older hardware.

DESIGN · 10. April 2018

Coherent Creature Design

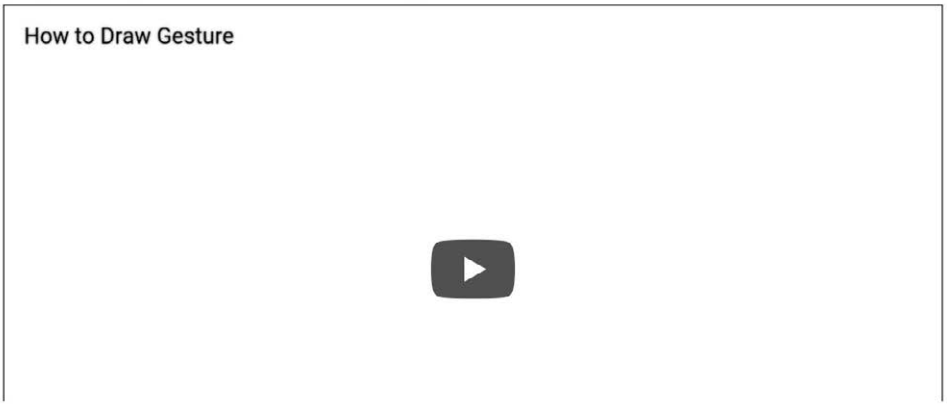
Designing the game world's creatures

As I promised in this chapter I will dig deeper into the designing process around the creatures that will be walking on the procedural terrain. Therefor I will present a technique, which is actually more of an artist's life hack to keep yourself unintentionally creative than an actual technique in my opinion. To elaborate further on this, I made a tool which helps me to stay coherent in the way I design environments and creatures in this particular project. I'm referring also to two design books I've read through very rapidly.
So, grab a pen and paper – we're be exploring the unexpected.

Gesture drawing

I bet those of you who are into drawing experienced this also once or twice while working on a piece. One is sketching a scene or character concept very rough and with a lot of gesture on a piece of paper (or digitally) but in the following steps, when it comes to outlining the shapes, the sketch becomes sort of stiff and loses its dynamic. The precise placement of the dark ink lines becomes the opposite of the strongly dynamic gesture you started working on. I realised pretty soon, when I did comics and cartoons, that I lose a lot of soulful characteristics and energy of the sketch when I "worked the illustration out". Also, I realised some years ago when we did life drawing in class that I produced way more vivid results when we did ten second gesture drawing compared to a 50 minutes anatomy study. I guess it was not so much the time span, but the freedom of not being able to work the drawing out within my comfort zone. To capture the pose with a few fast and vibrant brush strokes or splatter watered ink without having full control of the end result, generated new and interesting shapes I could take as a base for later refining. In the following weeks, I started to investigate into gesture drawing in general to pimp my stiff handed illustrations.

The following video by Stan Prokopenko is a cool point to start with.

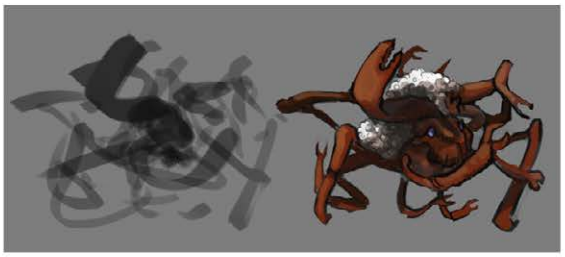


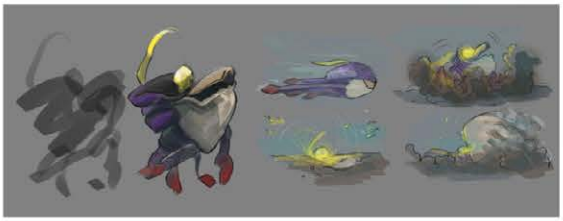
I still benefit from this change of mindset. Even when I do my favourite type of illustrations with a high, let's say, "richness in detail" I start with very intuitive gestures for some unexpected spice.

Blind drawing

To increase the effect of not controlling one's base sketch too much, I also began to draw with closed eyes. Sounds a bit weird, but I think everyone did this as a kid. I realised this for myself when trying to make a character for my game and the traditional way of setting up a skeleton (the sketches in the white box of the image) just gave me over and over again similar results. The other characters are made by starting with a blind drawn shape.

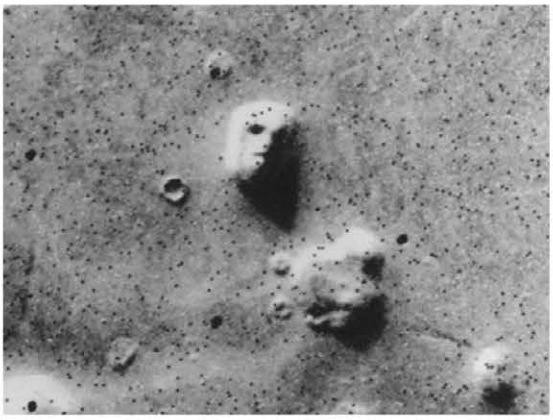
As you get better in drawing, I assume you will also get better in imagining things in front of your inner eye. In my opinion, and I am not a psychologist or neuro-scientist at all, the eye-hand coordination is kind of a brain-hand coordination that works also with closed eyelids. The tricky thing here is, if you train it well and the object or thing you imagine is not to complicated, you still get a decent and not totally unrecognizable result. But in this case, I rather get something totally out of any context so I can interpret forms and gestures into it. So, don't think of a dog when drawing a dog with closed eyes – just scribble stuff and try to find a suiting shape. Here are some more examples of these "fluid associations":





This associative approach is embedded in the human perception of the world. The effect that causes the "seeing faces in clouds" thing is called Pareidolia. It describes the phenomenon that people tend to see faces, or a better term would be "patterns", in structures where are none.

A face in the clouds?



E.T. is it you? This picture from a semi cratered highland on Mars shows simply a rock formation.

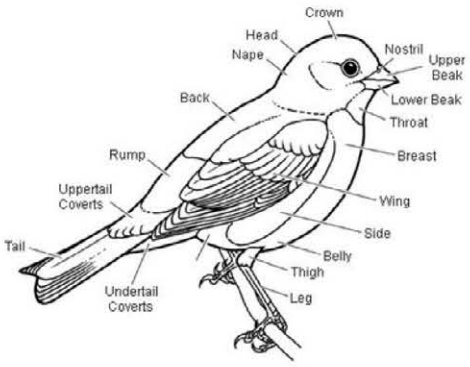
I really like this approach because it opens a wide variety of interpretations on just one starting layer which can be elaborated more in detail later on. Further it's not limited to visual appearances. As Wikipedia says also hidden messages in sound files that are played reverse or voices from random wind noise belong in the same category of slightly schizophrenic human interpretation.

Online available tools

I found the following website, al.chemy.org, as a reference for concept artist which exactly belong to the topic above. I downloaded it just a few days ago to try it on my drawing tablet. It works properly, even if the pen strokes are translated a bit obviously sometimes – but this might be caused by not enough variables in my settings. I can imagine this as a great tool for artists of all kinds, even when it focuses on digital painting and concept art. It even tries to include dynamics by extruding the line waves. I suggest you try it out yourself. This image is made with the provided software.

I downloaded it just a few days ago to try it on my drawing tablet. It works properly, even if the pen strokes are translated a bit obviously and very flat most of the time – but this might be caused by not enough variables in my settings. I can imagine this as a great tool for artists of all kinds, even when it focuses on digital painting and concept art. It even tries to include dynamics by extruding the line waves. I suggest you try it out yourself.

Excursion: Categories and ideal types



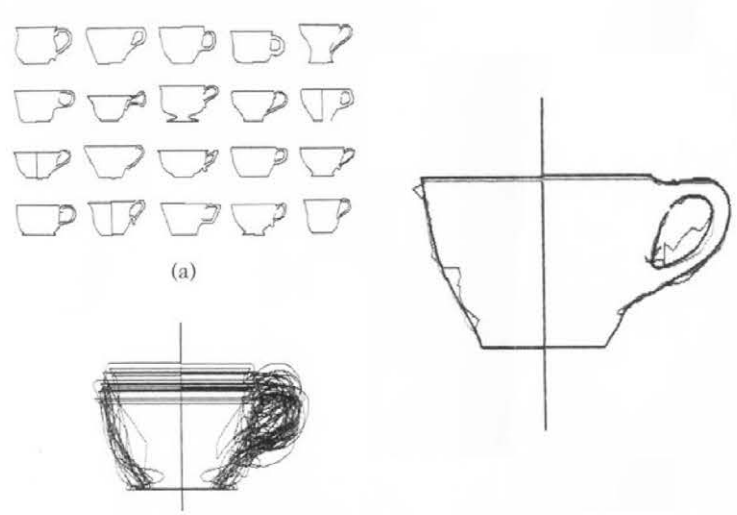
In a book called "Die semantische Wende" by Klaus Krippendorff I've found an interesting chapter about Carl von Linné, a Swedish scientist from the 18. Century that used to classify our living environment. We assume that the humans started very early to classify plants by their role for their survival. Some were eatable, others not. Some others that grow solid wood trunks can be used to keep fire burning. To classify the environment was, and still is very important for us to survive and is embedded in our instinct. Prejudice discussion opened :P When we think spontaneously of a certain animal, a bird for example, we barely have a certain colour in mind. Neither does the bird have a long neck like a heron nor does the bird has webbed feet. We rather think of an ideal or prototype bird like this one over here from a kid's science book.

Ideal type doesn't mean necessary "the best possible phenotype". It's more a term describing the most common shape of something in people's mind. This kind of bird shape represents the "default version" of any bird in western culture. Wings, not to small, not to big, medium long tail feathers, short neck, medium sized beak, thin legs with tiny claws. All the special features of other bird species are recognisable for us, because they differ from this ideal type. I guess that's also why it's easier to name a swan (or even just the silhouette) with all the extra shaping (long S-curved neck, huge white body, webbed feet) for children than to name a Phoenicurus, which is very close to the ideal of a bird.

To create unique, or at least not very default looking bird shapes (in the context of art), blind shape drawing and associative structure recognition (these scientific terms tho) are in my opinion good ways. They increase the

chance for not landing automatically in an ideal type of the desired creature. I also tried some morphing with a first sketch of a creature's head I didn't like much. It was way to unspectacular for what I expected it to be. I used the smudge tool in Photoshop and simply played around with it – way to controlled if I look back now.

But what I got: Several new base shapes of a given structure (eyes, mouth, jaw) which I could turn into another creature within the same style easily. Morphing is also described in the book same book from Krippendorff as a way of elaborating the common shape associated with things. The example shows how layering all shapes of the different cups make out the ideal type for "cup". I scanned the following image from the book. I put the credits on the end of the post.



Gunther Rehfeld, author of the german book "Game Design und Produktion" from 2014, mentions the importance of shapes in his chapter about concept art. He writes about how players must be able to determine if a creature is a friend or enemy just by its shape and from long distance. The same is valid for weapons of all kind.

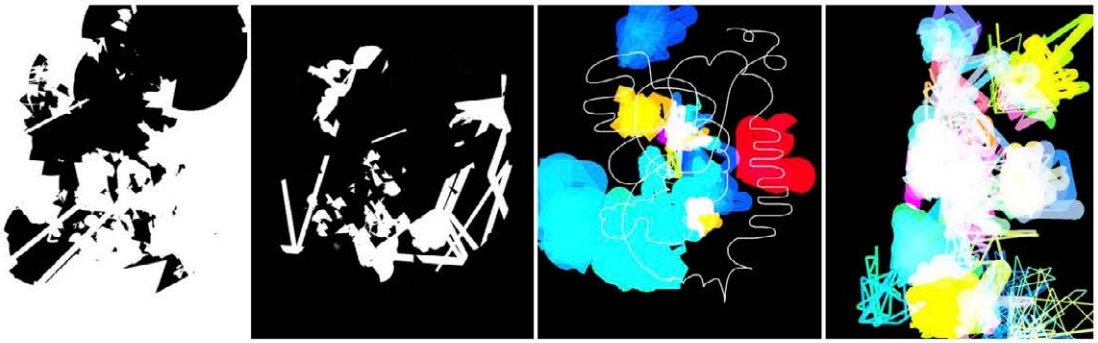
Coherence in creature and environment design

In order to use this great technique of blind drawing and its advantages over "just drawing monsters" I felt like I need a method behind all of this. One should be able to explain why a design appears the way it is. Like with the layering of versions to elaborate a default shape. Saying "I draw this church every day at the same time from my bathrooms window for one year to study lighting" is a well-made choice and method. But if you do it one day

only with ink on paper but the next day with crayon on wood and then with a stick into sand it becomes a bit random and weakens the system.
What I want to clarify here is: I wanted a concept and an as much coherent methodology as possible behind it. Traceability is the buzzword. And I claim this is key in any "scientific" approach as well as on a master thesis. So, generating random shapes and fill them out from associations is not enough. My personal solution was to program my own shape generator, based on values from the terrain generator. This way, the creatures I interpret from the shape are based on the same ruleset as the terrain of the world they live in.

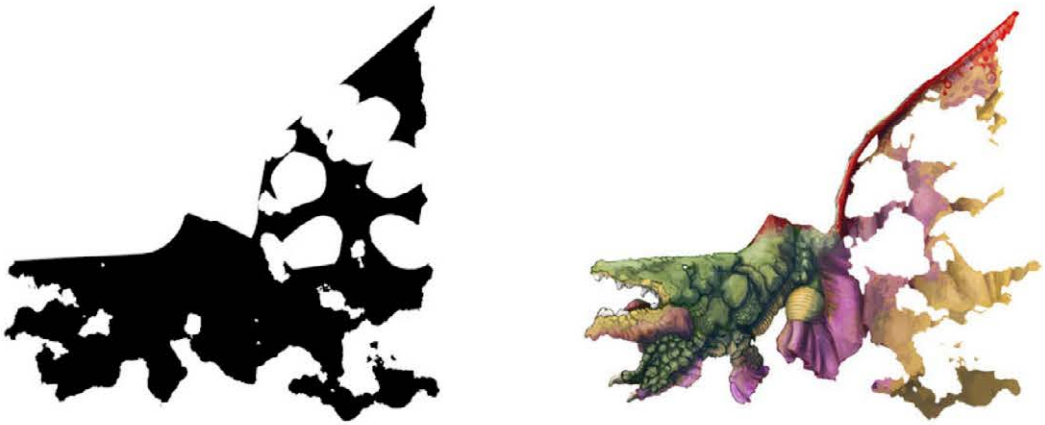
Shape generation tool

The application I made is a shape generator. It uses Unity's LineRenderer and TrailRenderer component to generate randomized shapes within set value borders. This examples here are not valid ones for my particular project, but still worth a peek. I used to have randomized colours and a randomized line thickness. On the third image, I tried to programed a draw mode option to directly edit the shapes in the generation process.



In the end, the black and white shapes from gave me the most satisfying results.
The starting shape is a perlin noise that has a circular gradient applied on it. It's basically the falloff generator from an early episode of Sebastian League's procedural landscape generator. The goal is to generate shapes this way that are directly dependent on the e.g. the noise seed of the landscape generator. Also, the decisions on spawn heights for trees and the water level influence the shape in a certain manner. I also considered physics based placement of rocks from the last post as a form giving method. All the shapes are then saved as png-files inside a folder on the desktop. Here some results I like.

With this base, I further elaborated the shape by adding colours and details up to a finished creature concept. One first example is a crocodile-like monster that lives on the beaches of the alien world. It may wave its flag tail around to attract other individuals or to mark its territory. I'm open for inputs.



Why not use genetic rules for shapes?

I want the creatures to fit in the world visually. The reason I don't want to hop on the train of designing them like they could exist on this planet is that I don't have enough capability to do all the checks for plausibility. In 3D modelling software, there is no limit to what you can do. A designer can create an elephant-sized turtle with legs thin as the ones of a flamingo. When I decide to include "biological correct evolution" into my project I would need to dig way deeper into science. Every decision on the design of the creatures then must be depending on these elaborated rulesets. Some games like the Swiss game Niche work with genetics to get to a coherent appearance. In my case I decide the body structure of each creature based on the generated shapes and colour scheme of the terrain.

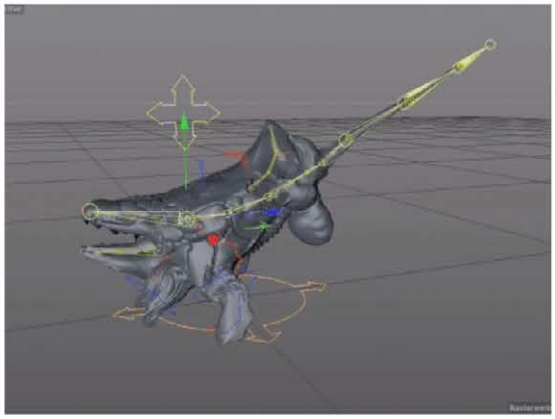
Concept art

The following images are the further elaborated shapes from the shape generator. The shapes as a coherent base for the artist's own interpretation is working quite well so far for this project and it's also a whole lot of fun. I am positive that the shapes bring up design solutions I wouldn't have discovered when constructing the skeleton of the creatures "traditionally". On the same time, while working on them, I again noticed how I fell back into designing animals I already know from memory. This could be a point for critique I am aware. But as creativity doesn't mean "innovation only" but also reinterpretation and collage and so on this is okay. A bigger issue was somehow the fact that I always saw mouths, eyes and legs in the shapes. I even realised shortly after my decision on which shapes I want to work on that I might have chosen them unconsciously on these criteria. To not be bound to it I picked the third creature (the bug like one) randomly from the folder. Here are the examples I made.





Translation into 3D



As I mentioned in my first post I use Cinema4d as my preferred software when it comes to 3D modelling. I admit having trouble with exporting textures and stuff sometimes in the past, but at the moment I think it does its job quite well. One thing I had to keep in mind was again the number of polygons I want to use for each model. This is also the reason why the models are not super smooth. I simply want to be able to work on proper conditions in Unity on my machine later on. The sculpted model was then exported as a fbx-file before I constructed the fish-like joint skeleton. The reason I exported it first was that I didn't want to have the sculpt tag on the object as I knew from previous projects that it might cause some issues with UV maps later in Unity. An easier step would have been to simply transform the sculpted mesh into a new object in Cinema4d. The result keeps the same. I rigged the mesh according to the creature's anatomy and tweaked around the settings for a while.

The upper image is a rapidly made rendering in Cinema4D. I didn't put much effort into it. I just wanted a first impression of a possible in game view.

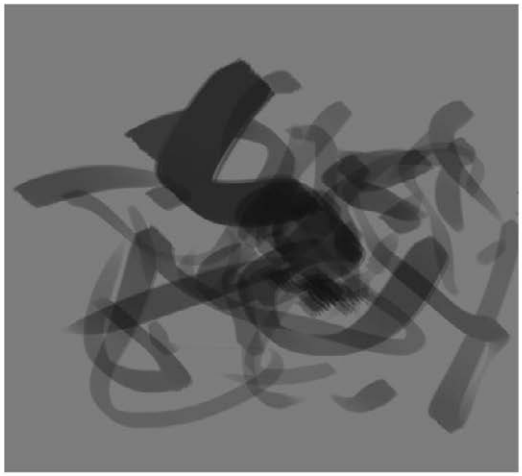
Critique helps

Yesterday I presented the current state of the project to my class and I got really good feedback from two of my classmates.

A girl, she does mostly graphic design, asked me **why I think everything must have a high traceability in design decisions.** I realised for myself that there is a break in this thinking as soon as the designer or artist starts to work on the shapes. The base is as possible (same algorithm that creates the noise pattern of terrain and so on) but what happens to the shapes after - how the creatures will look in the end - is open for the individual design decisions of the person working on it. I now want to point that particular thing out, as I think, in times of No Man's Sky and other games that use procedural generation to its finest, **it is still the designer's choice that makes the game stand out in the end.** The human, professional decision on aesthetics and stylization makes the design appealing, not any of those seed numbers from code. The worst would be if a design in the end looks arbitrary. And that's still a big issue with procedural generation for visual content.

When I was at Ludicious game festival in Zurich last January, I asked David O'Reilly, a great artist and animator who started working on games as well, how procedural generation influenced his work - I was mentioning Everything. As O'Reilly even wrote kind of a manifesto, it's actually an essay, that you can download from his website and in which he proclaims "coherence" as the most valuable criteria. Which I agree on very strongly. His answer was that he yet doesn't know how to cope with all the possibilities of procedural generation within a high grade of coherence, as the complex algorithm might always spit out arbitrary content in a way where the humans, at least in design, still must be the one instance of controlled filtering for valid, visual appealing results.

The other big critique came from an industrial designer who simply said he likes the blind drawn ones more than the procedural shapes. He highlighted the fact that I used the same side view of the creatures in the shapes I created by code. And he was damn right! I realised it and began to think about this. I guess the opacity of the brush strokes made the not-procedural shapes more three-dimensional. When comparing these two examples it becomes obvious how flat the procedural shape is, compared to the one I've drawn by hand:



I will start working on this problem with the next shape creature I create.
With this I close this topic of creature design for now and hope I could give you some references and insights :)

10.3 Story

Himmelfahrtskommando Jahr 6372 – Exosphäre Kepler-69c im Sternbild Schwan.

Leben - Danach suche ich. Ironisch angesichts des künstlich blau leuchtenden Neuro-Interfaces des Raumschiffs, der endlosen Dunkelheit hinter mir und nicht zuletzt wegen mir selbst, das ich mehr Binärcode als Fleisch und Blut bin. Seit 4000 Erdjahren nun schiesse ich mit einem Viertel der Lichtgeschwindigkeit im totstillen Vakuum zielstrebig in eine Richtung. Heute wurde ich schliesslich durch die Gravitation des Riesenplaneten aus dem Kryo-schlaf geweckt. Allein unterwegs zur letzten Chance. Kepler-69c.

Die Human-Embrionalstammzellen sind sicher verwahrt im „Bauch", einer Biobank auf halber Strecke in Richtung meiner Reiserute stationiert. Der Bauch wartet darauf einen Planeten mit geeinter Atmosphäre und Ressourcen anzusteuern. Keppler-69c erfüllt diese Kriterien. Unsere Intelligenz hat durch Spektralanalysen bereits genug Infos über die chemische Zusammensetzung des Planeten gesammelt, jedoch keine, über möglicherweise darauf existierende Organismen.
Eine weitere missglückte Kolonialisierung wäre der Untergang des letzten biologischen Humanoiden-Erbguts. Die bisherigen Versuche schutzanzugunbedürftige Humanoiden auf extraterrestrischen Planeten auszusetzten und eine gesunde Population aufzuziehen scheiterten. Sie wurden entweder durch Mikroorganismen, welches ihr Immunsystem nicht erkannte und deshalb nicht bekämpfen konnte, zu stark dezimiert oder sie zerstören durch ihre Intervention in die extraterrestrische Umgebung ihre Lebensgrundlage in wenigen Generationen selbst. Hätte man die Gefahren gekannt, wäre es ein Leichtes gewesen, die Stammzellen genetisch daran anzupassen und das Gleichgewicht des einheimischen Lebens nicht zu zerstören. Bei Keppler-69c soll dies nun in einem letzten Versuch angegangen werden. Indem die potentielle neue Heimat akribisch untersucht und ihre Lebensformen analysiert und klassifiziert werden.

Das ist meine Aufgabe. Die Rettung der letzten vollbiologischen Humanoiden und die Bewahrung einer fremden Vielfalt, die unsere irdische möglicherweise weit in den Schatten stellt.

Das Raumschiff setzt zum Eintritt in die tieferen Schichten der Atmosphäre an. Sobald die Landestelle vom Licht Kepplers Stern erwärmt wird, setze ich auf.

Dicker Nebel überzieht das Terrain und man sieht kaum die Hand vor Augen. „Stapf, stapf..." – leicht in den morastigen Boden eingesunken bemerke ich, wie unsicher man nach tausenden von Jahren im Eisblock auf den eigenen Beinen steht. Die Proteinstrukturen sind zwar seit Tagen, in der das Schiff noch zwecks topografischer Analysen um den Riesen kreiste regeneriert, jedoch dauert es etwas bis die KI das Raumanzugs-Gleichgewichtssystem von dem, des richtungslosen Alls an das der Gravitation Keppler-69c's anpasst. Trotz der Updates meiner Ausrüstung während meiner interstellaren Reise, die wegen der immensen Distanz zwischen dem Raumschiff und dem Host im „Bauch" irgendwann sowieso unterbrach, ist sie wohl bereits ein Dinosaurier. Falls noch Humanoide auf der Erde am Sys-tem forschen, hätten sie in der Zeit wohl ein Raumschiff konstruiert, das mich eingeholt hätte.
Ja – falls noch Humanoide auf der Erde leben. Wahrscheinlich sind sie längst digitalisiert und in den „Bauch" integriert worden. Vielleicht wurde nach meiner Abreise auch ein nähergelegener Planet mit erfülltem Kriterienschema entdeckt und der „Bauch" hat seine Mission längst erfüllt. Das kann mir hier niemand sagen. Jedenfalls muss ich meine Aufgabe wahrnehmen und erforschen, was es zu erforschen gibt. Ich habe das Wissen, mit den hier vorhanden zu sein prophezeiten Ressourcen einen Sender zu bauen, der die Anreise des Bauchs initiiert. Dieser wäre, sofern nach meiner Abreise noch daran weiterentwickelt wurde, hoffentlich in weniger als 2700 Lichtjahren hier. Aber noch bevor ich diesen Gedanken an die Vergangenheit und mein Verständnis von Zeit fertigspinnen kann...

..läuft mir genau so eine über den Fuss.

Oder ist es doch keine Spinne? Es hat drei stelzenartige Beine, eine Kruste mit sporadisch auftretenden, stacheligen Auswüchsen - und scheint sich nicht weiter für mich zu interessieren. Schnell zücke ich den „Genographen“ und schiesse ein Bild der Kreatur. Der Genograph analysiert die Molekularverbindungen des Wesens und spuckt sie zusammen mit dem vollen genetischen Code und einer Datei für das Archiv aus. Es scheint aus Kohlenstoffverbindungen zu bestehen, aus denen auch Erdorganismen aufgebaut sind. Das Vieh bewegt sich langsam, aber ohne auch nur einen Millimeter einzusinken über den wabbligen Grund. „Die Evolution hat nicht nur auf der Erde kreative Antworten auf die unterschiedlichen Umweltbedingungen gefunden“, schmunzle ich, als mir Bilder der alten Heimat wie als störender Glitch auf dem HUD meines Anzugs gesetzt werden. Die Systeme sind wohl wirklich etwas eingerostet. Ich katalogisiere das Wesen und gebe ihm einen Namen. Noch ganz fasziniert von meiner ersten Begegnung überrascht mich ein heftiger Windstoss. Der Nebel lichtet sich und nachdem das visuelle System die Blende richtig ein-gestellt hat, sehe ich es.

Ein Land - so weit und lebendig wie ich es nur aus Geschichten über meine alte Heimat, vor ihrer Zerstörung kannte. Urtümliche Schwammwesen schweben am Himmel, Grasähnlicher Wuchs und Farne bedecken die Hügel und mit jedem Zucken meiner Nerven und mit jedem Update Call meiner Systeme empfange ich mehr elektromagnetische Wellen von lebenden Organismen. Um mich herum eine Vielfalt so endlos wie das Land auf dem sie wandeln. Langsam begreife ich:

Das Abenteuer hat gerade erst begonnen. Und es wird riesig.

10.4 Anonymisierte Evaluationsbögen

Die Antworten wurden für die Dokumentation wörtlich übernommen.

Frage A: Beschreibe die Welt, in der das Game spielt:
Person 1: Kleinplanet mit Flora und Fauna. Erinnert an Tiefsee oder Traumsequenz.
Person 2: Verlassener Planet auf dem spezielle Tiere wohnen.
Person 3: Sonnenuntergang – Farben, Hügel, Wald. Alles ein wenig dunkel und suggeriert eine kleine „Welt“.
Person 4: Ein ferner Planet mit Bergen, Gewässern, Bäumen und Bodenvegetation. In der Ferne sieht man andere Planeten.
Person 5: Fremde Welt mit fetzeligen Tieren und unbekannten Pflanzen. Die Farben erinnern an ewigen Sonnenuntergang.
Person 6: It’s colourish, trees, leaves, forest, saturated colors, hilly. It’s nice that it is hilly. It makes you want to see what’s on the other side oft he hill. It feels a bit radioactive but in a calm, cosy way.
Person 7: Colourful, trippy, alien-like, seaside/mountain-range, many plants.
Person 8: Alien, multicolour, intriguing.
Person 9: No Man’s Sky. Romantisch, spacig. Mir gefällt’s.
Person 10: Stimmungsvoll, angelehnt an die Realität jedoch farbiger und formell fern. Träumerische Stimmung.
Person 11: 90er-Sytle-Sci-Fi Game. Etwas von Halo 1.
Person 12: Erdähnliches Terrain. Fühlt sich etwas orientierungslos an (Horizont / Handmarks würden helfen). Eher eng.

Frage B: Wo siehst, hörst, spürst du Ähnlichkeiten zwischen Spielelementen?

Person 1: Grafik + UI sehr kohärent und realistisch. Texturen. Kombination von 2D (Flächen) + 3D bei den Tieren. Korallenartige Pflanzen.
Person 2: Beim Umfeld, Bäume, Büsche, Gewässer. Die Abstimmung der Farben. Das Spiel hat eine schön abgestimmte Ästhetik. Dem Umfeld und den Tieren.
Person 3: Texturen sind ähnlich vom „Planet", Charakter hat andere Texturen.
Person 4: Die Musik passt sehr gut zu der visuellen Atmosphäre des Spiels.
Person 5: Tiere und Pflanzen wirken einheitlich. Farben scheinen zusammenzuwirken. Schrift ist im „Sci-Fi" Stil und vermittelt „Zukunft".
Person 6: I didn't think much about the music. I guess it's a good think cause then it propably fits the world. The typo, visuals, voice felt well together.
Person 7: The plants had similar shapes, seem to be from the same language. Even though they had different colours, it fitted well.
Person 8: good flow of colours.
Person 9: Pflanzen nebeneinander.
Person 10: Pflanzenwelt passt zu Himmel, Musik zu Astronautencharakter
Person 11: Leider nein. Kreaturen funktionieren mit der Flora.
Person 12: Farbe – Top! Formen – Top!

Frage C: Was in der Welt passt visuell nicht hinein?

Person 1: Berge nicht ganz kohärent bezüglich Struktur der Gestaltung – Berge sind zu normal, zu ungestaltet
Person 2: Nacht ist etwas dunkel. Charakter dürfte noch etwas spezieller, ausgefallener designet werden. Zoom DNA-Scanner ist etwas unübersichtlich, evtl. mit Farben arbeiten.
Person 3: Ist alles Stimmig, Bäume sehen zu ähnlich aus.
Person 4: Ist mir Nichts speziell aufgefallen. Ich finde Menü / Umgebung / Charakter passen sehr gut zusammen.
Person 5: "Ich". Der Astronaut. Was aber korrekt ist, er ist ja nicht von dort.
Person 6: Can't think of anything. Perhaps that the character you are, feels much more detailed than the surroundings.
Person 7: The space-man / explorer and his companion.
Person 8: All fit, maybe control graphics.
Person 9: Pflanzen , die "random" umherliegen. Die Sauerstoffflaschen sind optisch sehr detailliert ausgearbeitet: Nutzen => Herleitung zu Funktion deines Charakters. (Kann man mit den Flaschen fliegen?)
Person 10: Die Gestaltung des Scanners könnte abgefahrener sein. Momentan erinnert er mehr an die 80er Jahre.
Person 11: Die Landschaft-Elemente könnte von der Erde sein und der Astronaut ist unserer Welt am Entdecken. Verschwinden von Horizont bei leichtem Kameraschwenken. Träge Kamera. Dass Kamera nicht von alleine folgt. Bodentextur / Lichtstimmung (Himmel/Grund) Flora. Visier vom Astronaut.
Person 12: "Fisch auf Land" Bewegung der Aliens vs. Atmospäre. Sonst – Top!

Frage D: Wo und warum hast du Begeisterung gespürt?

Person 1: Tiere beobachten (gleichzeitiges Schweben und Schwimmen / mich interessiert wie die Welt aufgebaut ist. Wie sie sich unterscheidet) – analysieren – Infos lesen. Gesamt-stimmung gelungen – auch durch Farbabstimmung. Raumanzug realistisch.

Person 2: Beim Scannen der Tiere => Gefühl sie eingefangen zu haben. Durchs Wasser Marschieren => nicht möglich in echt.

Person 3: Wenn es das erste Mal geklappt hat mit DNA scannen.

Person 4: Die ganze Welt hat eine gute, spannende Atmosphäre und nach dem ersten Sichten von Kreaturen will man unbedingt noch mehr finden.

Person 5: durch die Welt rennen.

Person 6: I had the most fun when exploring the world, seeing what kind of landscapes could be generated. Going over hills to see what there was. See if my character could walk in water.

Person 7: To climb and explore the mountain felt „abenteuerlich".

Person 8: I had fun chasing the wavy boi that looked like me at the club. It was fun to chase them around while scanning them.

Person 9: Nacht-Visuals. Licht an/aus machen. Verzeichnis von Tieren.

Person 10: Beim Erblicken eines Aliens. Unerwartet getarnt und herausfordernd beim Scannen.

Person 11: Neues zu entdecken.

Person 12: Klettern – Auf hohe Punkte steigen. Terrain erkunden, loslaufen und über nächste Bergkante schauen.

Frage E: Wo und warum hat dich das Spiel gelangweilt?

Person 1: Keine Langeweile. Bei längerer Spieldauer: gut wäre Wege (Richtungen), Zusammenhänge zw. Tieren und Pflanzen,

Person 2: Habe es zeitlich nicht genug lang gespielt. Nach einer Weile würde es aber wahrscheinlich langweilig werden. Ein neuer Planet und Tiere wäre aber spannend. Nur das Scannen alleine ist nicht extrem spannend, weitere Interaktionen mit den Tieren wäre cool.

Person 3: Man hat zu lange mit Herumlaufen. => Einfache Achievements am Anfang =>Dann komplizierter.

Person 4: Es gibt kein Belohnungssystem für Tasks, die man erledigt hat. Der Scannprozess könnte noch ein Geräusch machen, wenn fertig. Man fühlt sich verloren nach einer Weile, weil man keine neuen Kreaturen gefunden hat.

Person 5: Während dem Intro möchte ich mich umschauen.

Person 6: It bored me when there were no more "aliens" to analize, and when I felt I had seen the generated content before just a bit different. But then I sarted exploring what would happen if I jumped down a high hill => started exploring my character instead

Person 7: the moment I couldn't see further and where I'm heading to explore.

Person 8: After a little while you have kind of seen all the scenery, maybe it would be nice to add caves.

Person 9: Keine Story, Welchen Nutzen hat das "Tiere sammeln"? Unklare Tutorials nach der Erklärung passiert nix. Grund?

Person 10: Nach dem Scan habe ich mehr erwartet, auch verstehe ich nicht genau was die Funktion der Drohne ist.

Person 11: Zu sehr repetitiv und zu häufig dieselben Elemente ohne Variation. Schnell keine oder wenig neue Elemente zu entdecken.

Person 12: DNA scannen hat keinen „Flow" wie beim Rumrennen. Man wird „unterbrochen".

Frage F: Was in der Spielewelt fühlte sich unecht an? (Im Sinne von „Was zerstört die Im-mersion“)
Person 1: Eigentlich nichts. Am ehesten die anderen Planeten. Vielleicht die Bergregionen (zu glatt). Die Verteilung der Bäume fühlt sich etwas unecht an. Nirgends ganz dicht. (hab's nicht gesehen)
Person 2: Navigation vom Scannen => Joystick. Übergang der Musik => Mixen fade-in/out. Die Stimme ist extrem langsam und demotiviert.
Person 3: Kamerazoom / Egoshooter-Perspektive evtl. einfacher
Person 4: Springen: Die Kamera für die Bewegung ist sehr smooth und reagiert sehr schnell. Dadurch hat man das Gefühlt, es herrscht eine andere Schwerkraft auf dem planeten. Je-doch finde ich passt die Höhe vom Springen nicht dazu => könnte höher sein / schwebender. Wie wenn man auf dem Mond springt.
Person 5: Bewegung (Geschwindigkeit ist ok. Seitwärts nicht möglich). Kamerakontrolle abbt nach.
Person 6: Bugs. When the song changes to another. The lag when it generates new content. When the aliens don't react to you.
Person 7: The plants and creatures obviously, but not unreal in a negative sense.
Person 8: Camera movement.
Person 9: Höhen bewältigen (Block).
Person 10: Der Horizont welcher nicht komplett dargestellt wird. Teilweise abrupte Kame-rabewegung (Beim Richtungswechsel ohne Kamera / schwenken)
Person 11: Bäume und Steine sind keine Hindernisse oder versperren den Weg. Streifen im Hinter-grund. Plötzliches oder schlagartiges Erscheinen der generierten Welt (Berge, Steine, etc.) Astronaut und Laufgeschwindigkeit sind unterschiedlich.
Person 12: Fehlende Weitsicht stört Navigation. Kletter Flüssiger, runterspringen und ste-cken bleiben. Kreaturen – Mix aus Bekanntem (z.Bsp. Fisch auf Land, gemischt mit der Optik der Bäume) und Fiktio-nalem.

11. Literatur-, und Webquellenverzeichnis

Literatur:

Huizinga, Johan: Homo Ludens. Reinbek bei Hamburg: Reclam (2001)

Rollings Andrew und Morris Dave: Game Architecture and Design. New Edition. Berkely Kalifornien: New Riders (2004)

Santayana, George: The Sense of Beauty. Being the Outline of Aesthetic Theory. New York: Dover Pub-lications, Inc. (1955)

Rehfeld, Gunter, Schmidt, Ulrich (Hrsg.): Game Design und Produktion. München. Hanser Verlag (2014)

Tolkien, John Ronald Reuel: The Lord of the Rings. London: Allen & Unwill (1954)

(Gründer) Goodman, Martin. (Heute in Besitz von The Walt Disney Company): Marvel Entertain-ment, LLC (1939 mit Namen “Timely Comics”)

Eddington, Arthur Stanley: Space, time and gravitation : an outline of the general relativity theory. Cambridge: Cambridge Unitversity Press (1920)

Eddington, Arthur Stanley: Space, time and gravitation : an outline of the general relativity theory. Cambridge: Cambridge Unitversity Press (1920)

Togelius Julian, Shaker Noor, Nielsen Mark J.: Procedural Content Generation in Games. A Textbook and Overview of current Research. Springer (2016)

O'Reilly, David: Basic Animation Aesthetics (2009)

Tuan, Yi-Fu: Topophilia – A Study of Environmental Perception, Attitudes and Values. Englewood Cliffs. New Jersey. Prentice Hall Inc. (1972)

Bachelard, Gaston: Poetik des Raumes. Frankfurt am Main. (1957)

Web:

https://www.masterstudiodesign.ch
https://www.zhdk.ch
http://www.bfs.admin.ch
https://store.steampowered.com
https://www.nzz.ch
https://www.newyorker.com
http://pcgbook.com
http://www.levelbased.com
http://kunstunterricht.ch
https://www.duden.de
http://www.davidoreilly.com/downloads/
http://weblog.jamisbuck.org
https://us.battle.net
https://assetstore.unity.com
http://www.gdconf.com
https://www.gdcvault.com
https://www.rollingstone.com
https://unity3d.com
https://blogs.unity3d.com
https://www.youtube.com
https://dictionary.cambridge.org
https://de.wikipedia.org
https://www.heise.de
https://gamerant.com
https://www.inverse.com
http://www.wired.co.uk
https://github.com
https://www.pokewiki.de
https://www.naturkundemuseum-chemnitz.de
https://stephaniestutz.ch/projects/5651361
https://www.thinkwithgoogle.com
https://www.mnenad.com

12. Bildquellenverzeichnis

Bild 1: http://www.indieretronews.com/2018/04/pacman-preview-possible-work-in.html
Bild 2: https://venturebeat.com/wp-content/uploads/2016/02/civ-1.jpg?strip=all
Bild 3: https://www.gamestar.de/artikel/everything-das-spiel-in-dem-man-alles-sein-kann-jetzt-auf-steam,3313133.html
Bild 4: https://www.nintendo.de/Spiele/NES/Super-Mario-Bros--803853.html
Bild 5: https://geekandsundry.com/terraforming-mars-one-of-the-greatest-board-games-in-the-galaxy/
Bild 6: http://de.reddead.wikia.com/wiki/Datei:Rdr_torquemada_lookout.jpg
Bild 7: http://reddead.wikia.com/wiki/File:Rdr_el_matadero00.jpg
Bild 8: https://www.t-online.de/spiele/id_80533098/-the-legend-of-zelda-breath-of-the-wild-im-test.html
Bild 9: http://www.nintendolife.com/news/2016/07/shigeru_miyamoto_provides_assurances_on_pikmin_4_development + https://www.youtube.com/watch?v=2lnFUBi1duA
Bild 10: http://www.pvp-serverler.gen.tr/2015/06/minecraft-server-tanitimi.html
Bild 11: http://www.pvp-serverler.gen.tr/2015/06/minecraft-server-tanitimi.html
Bild 12: Eigene Darstellung (© Mihajlo Nenad)
Bild 13: Eigene Darstellung (© Mihajlo Nenad)
Bild 14: Eigene Darstellung (© Mihajlo Nenad)
Bild 15: Eigene Darstellung (© Mihajlo Nenad)
Bild 16: Eigene Darstellung (© Mihajlo Nenad)
Bild 17: https://wallpapere.wallpaperstock.net/desert-red-sand-dunes-wallpapers_w40858.html
Bild 18: https://www.researchgate.net/figure/Perlin-noise-pattern-represented-as-greyscale-image-left-and-the-resulting-terrain_fig1_274384740?_sg=8HwSqumNNWDZxNjvY5MrS_VImTyT7bjHC4w1N4TuM8oGt-Q1St0XbaRycSd7k4_0_o8URV5VZR2I12VRJM2wzQ
Bild 19: https://www.researchgate.net/figure/Seven-layers-of-stacked-Perlin-noise-patterns-left-and-the-resulting-terrain-right_fig2_274384740?_sg=E_NXtjnC_ZGLBhN8kqDSnRqA3uA_xDuPOenJijmOE5hu4cFqZ_zTLcfcVG200Py6crh2uaCJTakrPnrb986dGQ
Bild 20: https://www.nintendo.de/Spiele/NES/Super-Mario-Bros--803853.html
Bild 21: http://kunstunterricht.ch/cms/grundlagen/224-grammatik-der-bildsprache-formale-gestaltungsprinzipien
Bild 22: https://www.philipphauer.de/galerie/caspar-david-friedrich-werke/
Bild 23: Von http://www.gemeentemuseum.nl/collection/item/6496 http://www.lifeproof.fr/mon_weblog/2011/01/piet-mondrian-lartiste-qui-aimait-peindre-les-arbres-by-stefania.html, Gemeinfrei, https://commons.wikimedia.org/w/index.php?curid=37636478
Bild 24: http://www.mazegenerator.net
Bild 25: http://teambrg.com/diablo-3/uliana-set-dungeon-build-guide-mastery/
Bild 26: Eigene Darstellung. Screenshot Unity (https://unity3d.com)
Bild 27: Eigene Darstellung. Screenshot Unity (https://unity3d.com)
Bild 28: Eigene Darstellung. Screenshot Unity (https://unity3d.com)
Bild 29: Eigene Darstellung. Screenshot Unity (https://unity3d.com)
Bild 30: Eigene Darstellung. Screenshot Unity (https://unity3d.com)
Bild 31: https://blogs.unity3d.com/2015/06/15/making-of-the-blacksmith-concept-and-art-production/
Bild 32: https://blogs.unity3d.com/2015/06/15/making-of-the-blacksmith-concept-and-art-production/
Bild 33: https://blogs.unity3d.com/2015/06/15/making-of-the-blacksmith-concept-and-art-production/
Bild 34: Screenshot (https://www.youtube.com/watch?v=vxF1osPkplA)
Bild 35: Screenshot (https://www.youtube.com/watch?v=k6xRZK-EZQQ)

Bild 35: Screenshot (https://www.youtube.com/watch?v=k6xRZK-EZQQ)
Bild 36: Screenshot (https://www.youtube.com/watch?v=k6xRZK-EZQQ)
Bild 37: http://www.indieretronews.com/2012/11/elite-dangerous-space-sim-and-remakes.html
Bild 38: Screenshot (https://www.gdcvault.com/play/1024265/Continuous-World-Generation-in-No)
Bild 39: https://commons.wikimedia.org/w/index.php?curid=1324635
Bild 40: https://de.wikipedia.org/wiki/Kunstformen_der_Natur#/media/File:Haeckel_Blastoidea.jpg
Bild 41: Screenshot (https://www.gdcvault.com/play/1024265/Continuous-World-Generation-in-No)
Bild 42: https://gamerant.com/no-mans-sky-planet-creature-creation/
Bild 43: https://www.reddit.com/r/NoMansSkyTheGame/comments/4x1jkg/nsfw_spoiler_swagasaurus_rex/
Bild 44: https://theplaylist.net/criterion-adds-new-films-andrei-tarkovsky-robert-bresson-albert-brooks-july-slate-20170417/
Bild 45: https://www.gamespot.com/videos/the-unknown-antichamber-gameplay/2300-6403130/
Bild 46: Screenshot (https://www.youtube.com/watch?v=nfiS_PV3V-4)
Bild 47: https://wacom.com/en-ch
Bild 48: Eigene Darstellung (© Mihajlo Nenad)
Bild 49: Eigene Darstellung (© Mihajlo Nenad)
Bild 50: Eigene Darstellung (© Mihajlo Nenad)
Bild 51: Eigene Darstellung (© Mihajlo Nenad)
Bild 52: Eigene Darstellung (© Mihajlo Nenad)
Bild 53: Eigene Darstellung (© Mihajlo Nenad)
Bild 54: Eigene Darstellung (© Mihajlo Nenad)
Bild 55: Eigene Darstellung (© Mihajlo Nenad)
Bild 56: Eigene Darstellung (© Mihajlo Nenad)
Bild 57: Eigene Darstellung (© Mihajlo Nenad)
Bild 58: http://fractalfoundation.org/OFC/OFC-2-4.html
Bild 59: Eigene Darstellung (© Mihajlo Nenad)
Bild 60: Eigene Darstellung (© Mihajlo Nenad)
Bild 61: Eigene Darstellung (© Mihajlo Nenad)
Bild 62: Eigene Darstellung (© Mihajlo Nenad)
Bild 63: Eigene Darstellung (© Mihajlo Nenad)
Bild 64: Eigene Darstellung (© Mihajlo Nenad)
Bild 65: Eigene Darstellung (© Mihajlo Nenad)
Bild 66: Eigene Darstellung (© Mihajlo Nenad)
Bild 67: Eigene Darstellung (© Mihajlo Nenad)
Bild 68: Eigene Darstellung (© Mihajlo Nenad)
Bild 69: Eigene Darstellung (© Mihajlo Nenad)
Bild 70: Eigene Darstellung (© Mihajlo Nenad)
Bild 71: Eigene Darstellung (© Mihajlo Nenad)
Bild 72: https://suxeedo.de/glossary/storytelling-content-marketing/ und eigene Darstellung (© Mihajlo Nenad)
Bild 73: (© Mihajlo Nenad)
Bild 74: Eigene Darstellung (© Mihajlo Nenad)
Bild 75: Eigene Darstellung (© Mihajlo Nenad) nach nach Hunicke, LeBlanc, Zubek, 2004
Bild 76: Eigene Darstellung (© Mihajlo Nenad)
Bild 77: Eigene Darstellung (© Mihajlo Nenad)
Bild 78: Eigene Darstellung (© Mihajlo Nenad)
Bild 79: Screenshot (https://www.youtube.com/watch?v=uNa_Xps7HwE&t=1718s)
Bild 80: Screenshot (https://www.youtube.com/watch?v=uNa_Xps7HwE&t=1718s)
Bild 81: Screenshot (https://www.youtube.com/watch?v=t5ZaSPDwgo8)
Bild 82: Screenshot (https://www.youtube.com/watch?v=t5ZaSPDwgo8

Bild 83: https://stephaniestutz.ch/projects/5651361
Bild 84: Eigene Fotografie (© Mihajlo Nenad)
Bild 85: Eigene Darstellung (© Mihajlo Nenad)
Bild 86: Eigene Darstellung (© Mihajlo Nenad)
Bild 87: Eigene Darstellung (© Mihajlo Nenad)
Bild 88: Eigene Darstellung (© Mihajlo Nenad)
Bild 89: Eigene Darstellung (© Mihajlo Nenad)
Bild 90: Eigene Darstellung (© Mihajlo Nenad)
Bild 91: Eigene Darstellung (© Mihajlo Nenad)
Coverbild: Eigene Darstellung (© Mihajlo Nenad)

13. Ludologie

[1] Iwatani, Tōru: Pac-Man. Namco. CPU: Z80 (Arcade Machine). (1980 Japan)
[2] Meier, Sid: Civilization. Micro Prose (ab 1996). PC. (1991)
[3] O'Reilly, David: Everything. Play Station 4, Mac, PC. (2017)
[4] Miyamoto, Shigeru: Super Mario Bros. Nintendo. Famicom Nintendo Entertainment System. (1985 Japan)
[5] Fryxelius, Jacob, Fryxelius, Isaac: Terrafroming Mars. FryxGames, Schwerkraft-Verlag. Karten-, Brettspiel (2016)
[6] Cantamessa, Christian, Benzies, Leslie: Red Dead Redemption. Rockstar San Diego, Rockstar Games. Xbox360, Play Station 3 (2010 USA/Kanada)
[7] Aonuma, Eiji, Fijibayashi, Hidemaro: The Legend of Zelda: Breath of the Wild. Nintendo Entertainment Planning & Development. Wii U, Nintendo Switch (2017)
[8] Persson, Markus, Bergensten, Jens: Minecraft. Mojang, Microsoft Studios, 4J Studios, Other Ocean Interactive. PC, Android, iOS, Xbox360 und One, Raspberry Pi, Play Station 3 und 4, Play Station Vita, Windows phone, Nintendo Wii, Apple TV, Amazon Fire TV, Ninten-do Switch, Nintendo DS (new) (2009-2017)
[9] Toy, Michael, Wichman, Glenn, Arnold, Ken, Lane, Jon: Rogue. Apple II, Atari ST, C64, DOS, Linux, Mac OS (1980)
[10] Diablo 3. Blizzard Entertainment. PC, Mac OS, Play Station 3 und 4, Xbox 360 und One (2012 - 1014)
[11] Braben, david, Bell Ian: Elite. Acornsoft, Firebird. BBC Micro, Acorn Electron, Apple II, Schneider/Amstrad CPC, C64, Sinclair ZX Spectrum, MSX, Tatung Einstein, IBM PC, Amiga, Atari ST, Nintendo Entertainment System (1984)
[12] Murray, Sean: No Man's Sky. Hello Games. Play Station 4, PC. (2016
[13] Adams, Tarn, Adams, Zach: Dwarf Fortress. Tarn Adams. PC, Mac OS, Linux (2006)
[14] Bruce, Alexander: Antichamber. Demruth. PC, Mac OS, Linux (2013)
[15] Antoniades, Tameem, Metthews, Dominic, Ashman-Rowe, Elizabeth: Hellblade: Senua's Sacrifice. Ninja Theory. PC, Play Station 4, Xbox One (2017/18)
[16] Pokémon Snap. Nintendo. Nintendo 64 (1999 Japan)
[17] Ancel, Michel: Beyond Good and Evil. Ubisoft. Nintendo Game Cube, Play Station 2 und 3, XBox and Xbox 360 (2003)
[18] Stutz, Stephanie: Expedition Perm (2012)