

Prozedurale Terrain-Generierung mit zielgerichteten Verteilungsfunktionen

Procedural Terrain Generation with Targeted Distribution Methods

Hannes Klein

Bachelor-Abschlussarbeit

Betreuer: Prof. Dr. Christof Rezk-Salama

Trier, 27.7.2017

Kurzfassung

Im Laufe dieser Bachelorarbeit ist ein Editor zur prozeduralen Generierung von Karten entstanden, der in *Unity* eingesetzt werden kann. Zentrale Aufgabe dieses Editors ist es Inseln zu erschaffen und zu gestalten. Die vorliegende Arbeit dient als Dokumentation und Anleitung für diesen Editor.

Zuerst wird darauf eingegangen was die Grundidee hinter der Erstellung dieses Editors ist und was für Besonderheiten er bietet. Danach wird mehr auf die technische Umsetzung eingegangen und welche Algorithmen dafür verwendet wurden. Im Kapitel Editor wird schließlich die Benutzung des Editors im Detail erklärt und anschließend noch ein Ausblick mit Ideen für Erweiterungen gegeben.

Verfügbar ist der Editor auf *Github.com* [Bac].

Inhaltsverzeichnis

1	Einleitung	1
1.1	Spielkonzept	1
1.2	Grundidee	2
2	Noise	4
2.1	Normalisierung und Äqualisierung	4
3	Höhengenerierung	7
3.1	Poisson Disk Sampling	7
3.2	Kombination	7
3.2.1	Addieren	8
3.2.2	Potenzieren	11
4	Kartengenerierung	12
5	Editor	14
5.1	Controller	14
5.2	Island Creator	15
5.3	Island Distribution	16
5.4	Splatmap Creator	16
5.5	Map Analyser	16
5.6	Noise Creator	17
6	Zusammenfassung und Ausblick	19
	Literaturverzeichnis	22
	Erklärung der Kandidatin / des Kandidaten	23

Abbildungsverzeichnis

1.1	<i>Anno 1404</i>	2
1.2	<i>Minecraft</i>	2
2.1	Beispiel für Perlin Noise mit bis zu fünf Oktaven	4
2.2	Das unveränderte Histogramm einer <i>Noise-Map</i>	5
2.3	Das mit der Normalisierung veränderte Histogramm	6
2.4	Das mit der Äqualisation veränderte Histogramm	6
3.1	Als <i>Animation Curve</i> angegebene Kurve zur Bestimmung der Grundform der Inseln	8
3.2	Kurven mit einer Exponentenbasis von 2, 4 und 7	9
3.3	Kombination mit einer durchgehenden Gewichtung von 0,6	9
3.4	Kombination mit einer durchgehenden Gewichtung von 0,7	10
3.5	Kombination mit einer Gewichtung die von einer Kurve vorgegeben wurde	10
3.6	Kombination mit einem Exponenten der über eine Kurve definiert wurde	11
4.1	Die Karte mit unveränderter <i>Noise-Map</i>	13
4.2	Die Karte mit äqualisierter <i>Noise-Map</i>	13
5.1	Der Editor für den Controller	14
5.2	Der Editor der das Aussehen der Inseln verändert	15
5.3	Der Editor für die Inselverteilung	16
5.4	Der Editor für das erstellen der verschiedenen Gebiete	17
5.5	Der Editor für das analysieren der Höhenkarte	17
5.6	Der Editor für das definieren einer <i>Noise</i>	18
6.1	<i>Poisson-Disk-Sampling</i> beeinflusst durch eine <i>Noise-Map</i>	20
6.2	Die entstandene Graslandschaft entspricht zwar der erwünschten Größe, ist jedoch zu Hügelig um Gebäude darauf zu platzieren	20
6.3	In <i>Anno 1404</i> sind die Inseln größtenteils flach	21

1

Einleitung

Prozedurale Generierung ist das automatische Erzeugen von digitalen Inhalten mit Hilfe von pseudo-zufälligen Werten[Prob]. Diese Inhalte können Texturen, 3D-Objekte oder sogar Musikstücke sein, aber das Gebiet mit dem sich diese Arbeit befasst ist die Prozedurale Levelgenerierung. Diese ist bei Videospielen beliebt da mit ihr ständig neue Level erzeugt werden können, die das Spielerlebnis frisch halten.

Prozedurale Levelgenerierung kann sehr unterschiedlich aussehen. Ein 2D-Jump'n'Run hat ganz andere Ansprüche an das Level-Design als zum Beispiel ein Strategiespiel. Beim Entwickeln einer prozeduralen Level-Generierung muss darum zuerst festgelegt werden wie die Spielmechaniken in etwa aussehen und welche Voraussetzungen sie mit sich bringen.

1.1 Spielkonzept

Für den vorliegenden Levelgenerator existieren noch keine umgesetzten Spielmechaniken. Um allerdings ein besseres Verständnis davon zu bekommen, was mit diesem Projekt erreicht werden sollte wurde vorher ein grundlegendes Spielkonzept festgelegt.

Das Spiel ist ein Aufbaustrategiespiel, dass sich an der *Anno*-Serie orientiert(siehe Abb. 1.1). Man baut eine Siedlung und versorgt die Bewohner zuerst mit den notwendigsten Gütern wie Nahrung und Wasser. Um das Leben der Bewohner zu verbessern müssen diese mit besseren Gütern versorgt werden, die zuerst gefunden werden müssen. Diese Güter können Mineralien wie Gold oder Silber sein, aber auch Nahrungsmittel die nur in bestimmten Gebieten wachsen.

Ein wichtiger Aspekt ist also das Entdecken neuer Gebiete, insbesondere Inseln, auf denen neue Rohstoffe entdeckt und abgebaut werden können. Durch die prozedurale Generierung sind diese Gebiete jedes mal neu verteilt und bieten ein ständig neues Spielerlebnis.



Abb. 1.1. *Anno 1404*

1.2 Grundidee

Eine wichtige Komponente bei der prozeduralen Levelgenerierung ist die Rauschfunktion, oder *Noise*-Verfahren. Das wahrscheinlich bekannteste Beispiel hierfür ist *Minecraft*, in dem ein dreidimensionales *Noise*-Verfahren verwendet wird um eine theoretisch unendliche Landschaft zu generieren(siehe Abb. 1.2).

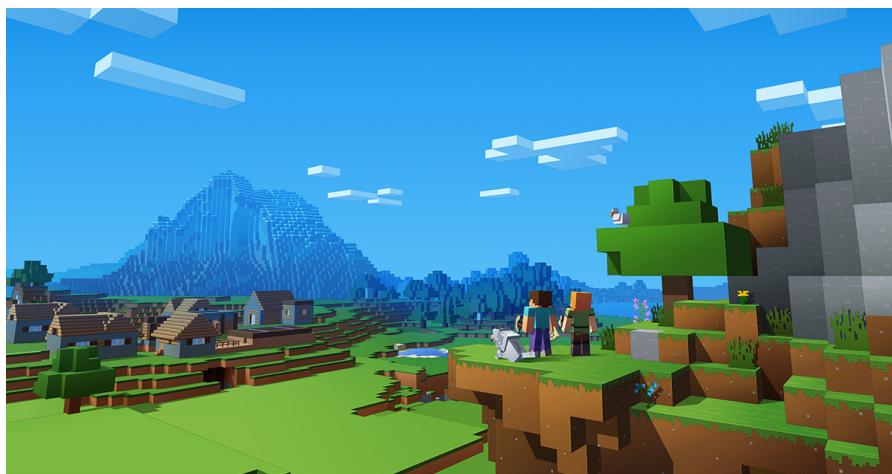


Abb. 1.2. *Minecraft*

Für dieses Projekt ist eine zweidimensionales *Noise* ausreichend, da keine Informationen für die Tiefe ermittelt werden müssen. Die erstellten Werte können als Höhenwert interpretiert werden um eine dreidimensionale Landschaft zu erzeugen.

Das Verteilen der Rohstoffe kann mit Hilfe einer weiteren *Noise-Map* ermittelt werden. Wenn zum Beispiel ein Berg entsteht kann anhand dieser weiteren *Noise-Map* festgestellt werden, ob dort Gold oder Silber abgebaut werden kann.

Das erstellen von Inseln ist mit *Noise*-Verfahren allerdings nicht so einfach möglich. Es können zwar Inseln entstehen, diese sind jedoch zu selten und unvorhersehbar.

Um also gezielt Inseln entstehen zu lassen muss zuerst ein Punkt festgelegt werden, der den Mittelpunkt der Insel festlegt. Danach wird jeder Punkt der *Noise-Map* abhängig vom Abstand zu diesem Mittelpunkt beeinflusst. Die benötigten Inselmittelpunkte werden im vorliegenden Beispiel mit dem *Poisson-Disk*-Algorithmus verteilt, da man damit einen Mindestabstand zwischen den Inselmittelpunkten festlegen kann[Poia].

Die Inspiration zu dieser Idee kam von einem Artikel auf der Seite **redblobgames.com** [Isl].

2

Noise

Als *Noise*-Verfahren wurde in diesem Projekt die *Perlin-Noise* verwendet. Diese wurde 1984 von Ken Perlin entwickelt und wird mit interpolieren von pseudozufälligen Gradientenwerten berechnet[Pera]. Für dieses Projekt wurde eine bereits existierenden Implementierung [Perb] verändert und in der Klasse *NoiseCreator.cs* gekapselt.

Bei den berechneten *Noise-Maps* wird im folgenden immer davon ausgegangen, dass die Werte zwischen Null und Eins liegen.

Entscheidend beim erstellen einer *Noise-Map* ist die Skalierung. Ein höherer Wert entspricht einem Heraus-zoomen und erhöht damit die Frequenz.

Eine Frequenz einer Rauschfunktion kann schnell repetitiv werden. Daher werden mehrere Oktaven mit steigenden Frequenzen aber sinkender Gewichtung zusammengelegt um eine detailliertere *Noise-Map* zu erzeugen (siehe Abb. 2.1)[Fra].

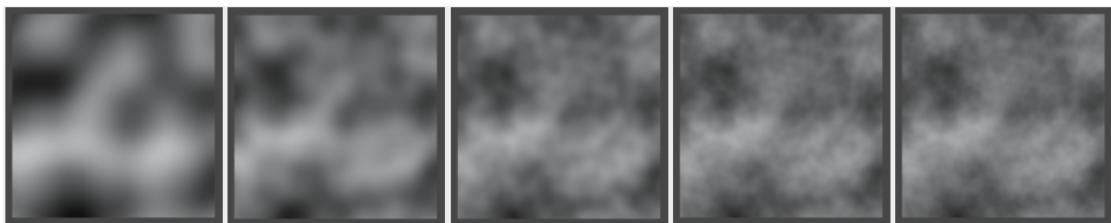


Abb. 2.1. Beispiel für Perlin Noise mit bis zu fünf Oktaven

Die entstandene *Noise-Map* kann noch mit einem Exponenten verrechnet werden. Wenn dieser Exponent größer als Eins ist, werden kleinere Werte dadurch weiter nach unten gedrückt. Dies kann dabei helfen Berge besser heraus kommen zu lassen.

2.1 Normalisierung und Äqualisierung

Ein Problem was sich bei der Erstellung der *Noise-Maps* ergeben hat, ist dass die meisten Werte im mittleren Bereich liegen und Extremwerte selten sind(siehe Abb. 2.2). Lösungen hierfür sind die Histogrammnormalisierung und die Histogrammäqualisierung.

Bei der Histogrammnormalisierung wird der größte und der kleinste Wert der momentanen Noise-Map verwendet und die Werte dann mit folgender Rechnung verändert[Nor].

$$I_N = (I - Min) \frac{NewMax - NewMin}{Max - Min} + NewMin$$

Das angestrebte Maximum beträgt im vorliegenden Fall Eins und das Minimum Null, weshalb die Rechnung vereinfacht werden kann.

$$I_N = (I - Min) \frac{1}{Max - Min}$$

Das Problem hierbei ist, dass nur diese Extremwerte beachtet werden und das Histogramm dadurch nur wenig verbreitert wird(sieh Abb.2.3).

Eine bessere Lösung stellt die Histogrammäqualisation dar[Aeq]. Hierbei wird eine kummulative Histogramm verwendet um die Werte neu zu verteilen. Obwohl dabei einige Wertebereiche verloren gehen sind die Werte insgesamt viel besser verteilt(siehe Abb. 2.4).

Warum diese Verfahren nützlich sind wird im Kapitel Kartengenerierung weiter ersichtlich.

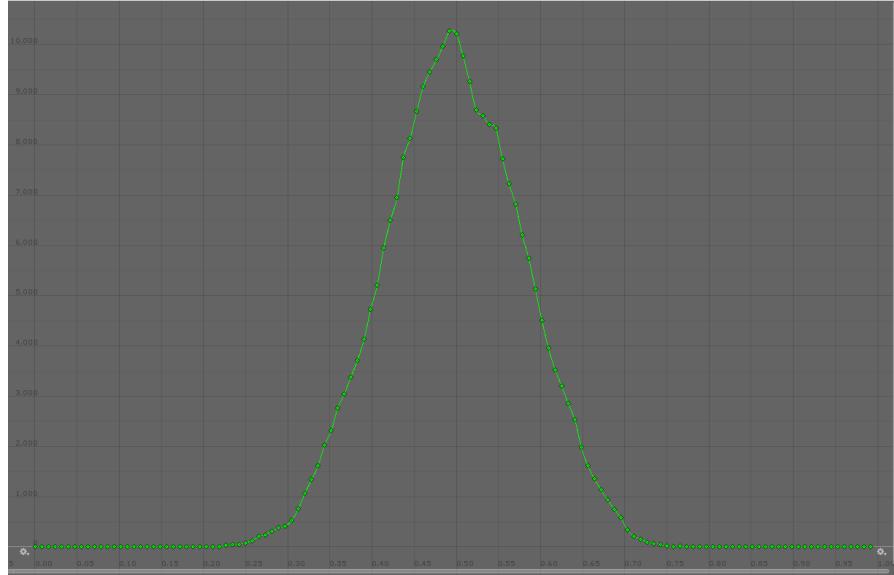


Abb. 2.2. Das unveränderte Histogramm einer *Noise-Map*

Alle vorgestellten Funktionen sind in der Klasse *NoiseCreator.cs* implementiert, die auch mit anderen *Noise*-Verfahren umgehen kann.

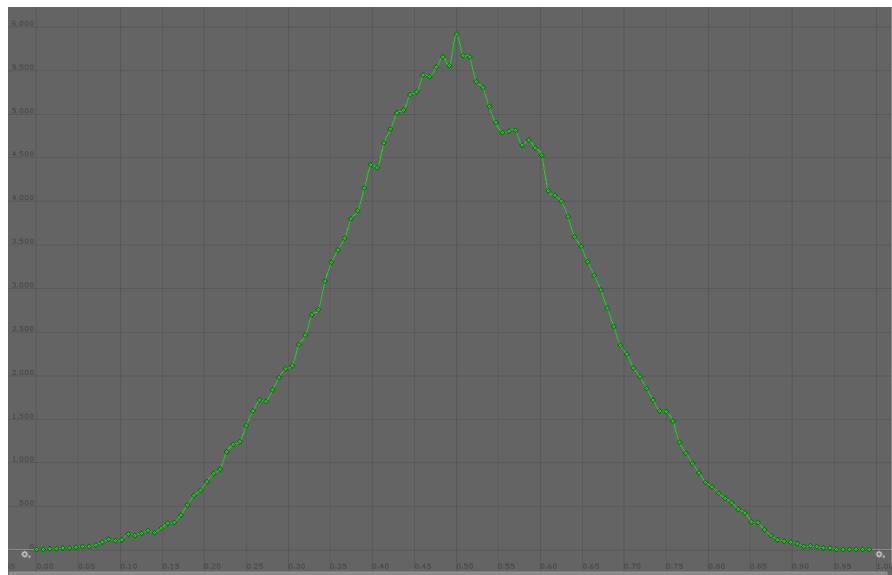


Abb. 2.3. Das mit der Normalisierung veränderte Histogramm

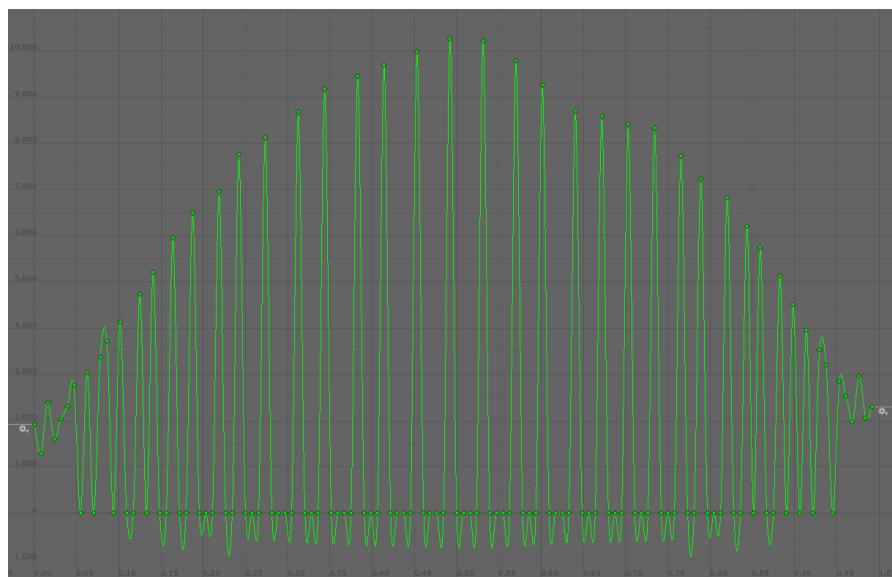


Abb. 2.4. Das mit der Äqualisation veränderte Histogramm

3

Höhengenerierung

Die Höhengenerierung ist ein wichtiger Teil der Kartengenerierung. Das hier erzeugte Höhenfeld wird später in der Kartengenerierung verwendet um zu entscheiden was für Gebiete entstehen.

Die Höhengenerierung besteht aus dem verteilen von Punkten mit Hilfe des *Poisson-Disk*-Algorithmus und dem Kombinieren mit einer *Noise-Map*.

Im Skript *Controller.cs* wird die grundlegende *Noise-Map* generiert. Außerdem kann hier das Generieren von Inseln ein und aus geschalten werden.

3.1 Poisson Disk Sampling

Beim *Poisson-Disk-Sampling* wird ein Feld mit Punkten aufgefüllt, die einen Mindestabstand von einander haben[Poia].

Dabei wird von jedem Punkt aus versucht weitere Punkte zu finden, die den Mindestabstand zu allen anderen Punkten einhalten. Mithilfe einer Gitterdatenstruktur kann die Kollisionsabfrage effizient durchgeführt werden.

Die Anzahl der Versuche die von jedem Punkt aus gestartet werden um weitere Punkte zu finden, sind entscheidend darüber wie voll gepackt das Feld sein wird, jedoch steigt dadurch auch die Rechenzeit.

In diesem Projekt wurde ein existierende Implementierung [Poib] abgeändert und in der Klasse *IslandDistribution.cs* gekapselt. Hier können optional die Punkte auch manuell angegeben werden.

3.2 Kombination

Die Kombination einer *Noise-Map* mit den im vorherigen Schritt verteilten Punkten ist der Schwerpunkt dieser Arbeit. Auch der Artikel von der die ursprüngliche Idee kam hatte keine zufriedenstellende Lösung für diese Aufgabe[Isl]. Zwei Lösungen wurden im Laufe dieser Arbeit entwickelt und in der Klasse *IslandCreator.cs* implementiert.

Zuerst muss jedoch für jeden Punkt auf der Karte die Distanz zu den Mittelpunkten berechnet werden. Dafür muss eine Distanz festgelegt werden, bis zu der die Insel noch Einfluss auf die *Noise-Map* hat, also sozusagen die Größe der Insel

angibt. Um die Weiterverarbeitung zu vereinfachen wurde die Distanz normiert indem sie durch diesen Größenwert geteilt wird.

Das heißt die Distanz liegt immer zwischen Null und Eins, wobei Null in der Mitte und Eins am äußeren Rand liegt. Punkte die noch weiter außerhalb liegen haben ebenfalls die Distanz Eins. Wenn ein Punkt in der Nähe zweier Mittelpunkte liegt wird nur die geringere Distanz beachtet.

3.2.1 Addieren

Die Distanz und die *Noise-Map* zu addieren ist nicht so einfach möglich, da sonst Werte über Eins heraus kommen könnten. Es ist also nötig eine Gewichtung anzugeben, die angibt wie stark die Distanz die *Noise-Map* beeinflusst.

Zusätzlich ist es praktisch die Distanz vorher zu bearbeiten, da in der Mitte meistens ein höherer Wert als Außen gewünscht ist. Mit einer von *Unity* vorgegebenen *Animation Curve*, kann eine Kurve angegeben werden, die die Grundform der Inseln vorgibt(siehe Abb. 3.1).

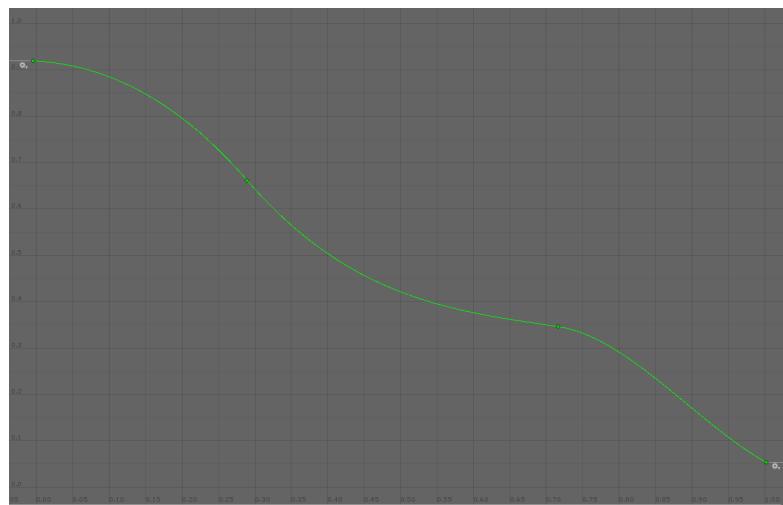


Abb. 3.1. Als *Animation Curve* angegebene Kurve zur Bestimmung der Grundform der Inseln

Optional kann auch ein Wert angegeben werden, der mit der negativen Distanz potenziert wird. Damit entsteht eine Kurve die in etwa der Kurven in Abbildung 3.2 entspricht.

Es hat sich herausgestellt, dass es problematisch ist nur einen Wert für die Gewichtung angeben zu können. Wenn der Wert zu niedrig liegt können kleine Inseln außerhalb der richtigen Inseln entstehen(siehe Abb. 3.3). Ist der Wert allerdings zu hoch sehen die Inseln sich zu ähnlich, da die *Noise-Map* nicht mehr genug hervortreten kann (siehe Abb. 3.4).

Die Lösung hierfür war es, die Gewichtung ebenfalls mit einer Kurve angeben zu lassen. Das heißt die Gewichtung ist ebenfalls abhängig von der Distanz zum

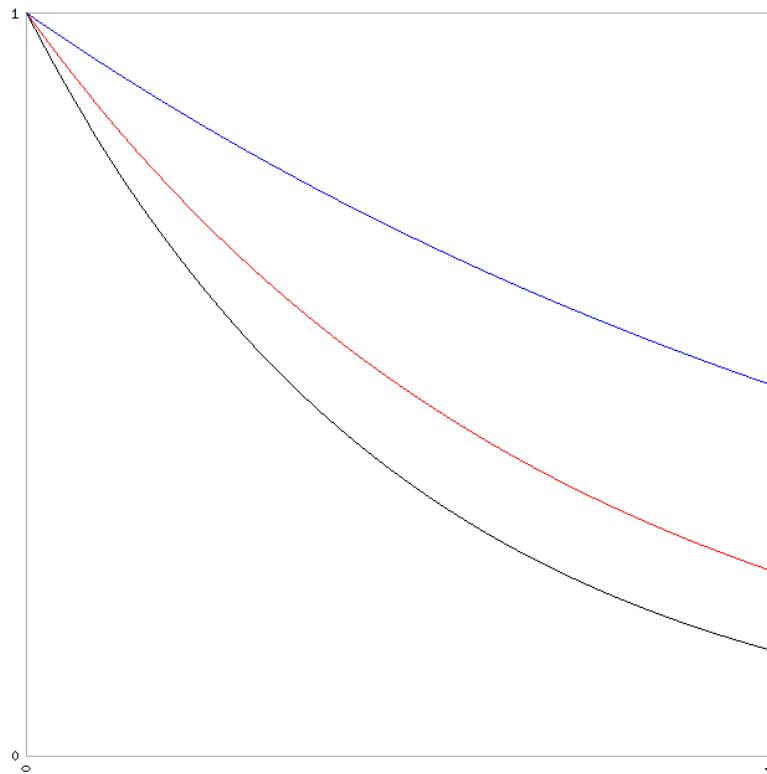


Abb. 3.2. Kurven mit einer Exponentenbasis von 2, 4 und 7

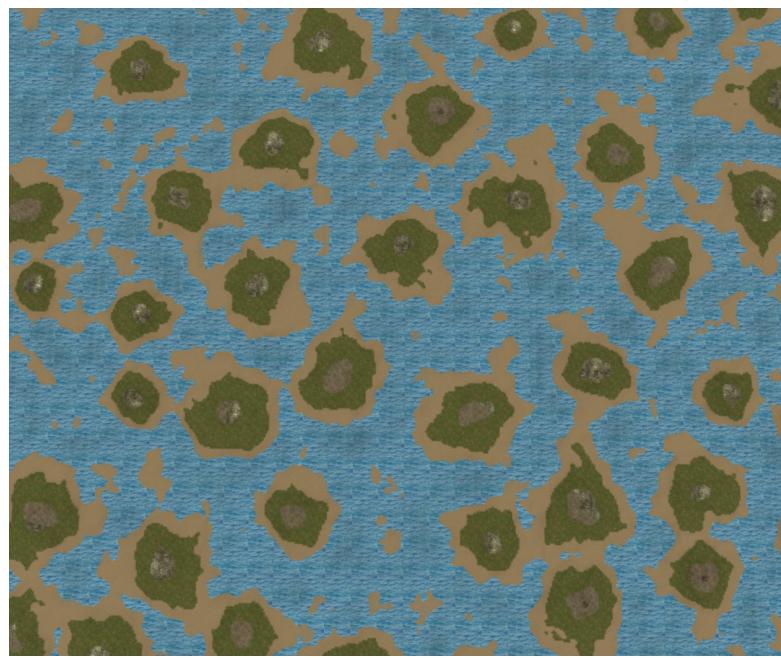


Abb. 3.3. Kombination mit einer durchgehenden Gewichtung von 0,6

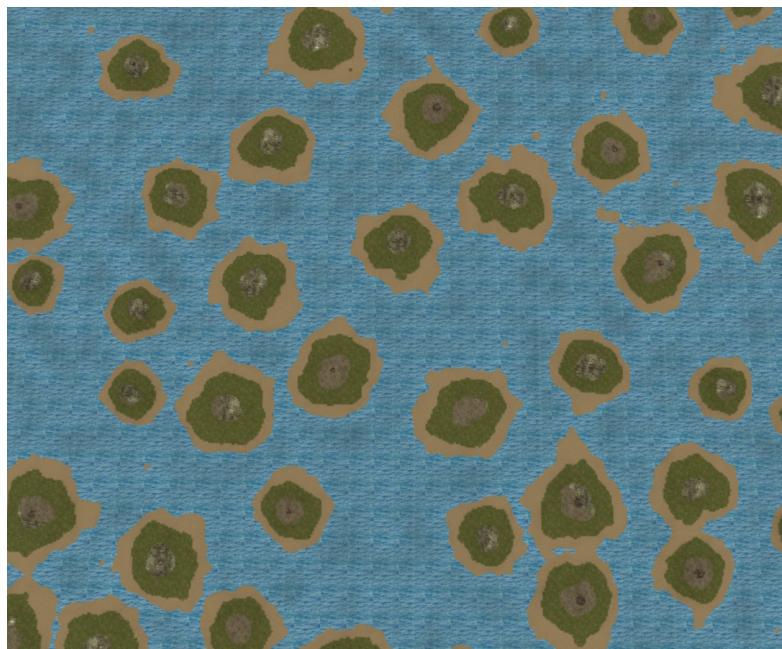


Abb. 3.4. Kombination mit einer durchgehenden Gewichtung von 0,7

Inselmittelpunkt. Damit kann man in der Mitte der Insel die *Noise-Map* besser hervor treten lassen und sie außen trotzdem gut kontrollieren(siehe Abb. 3.5).

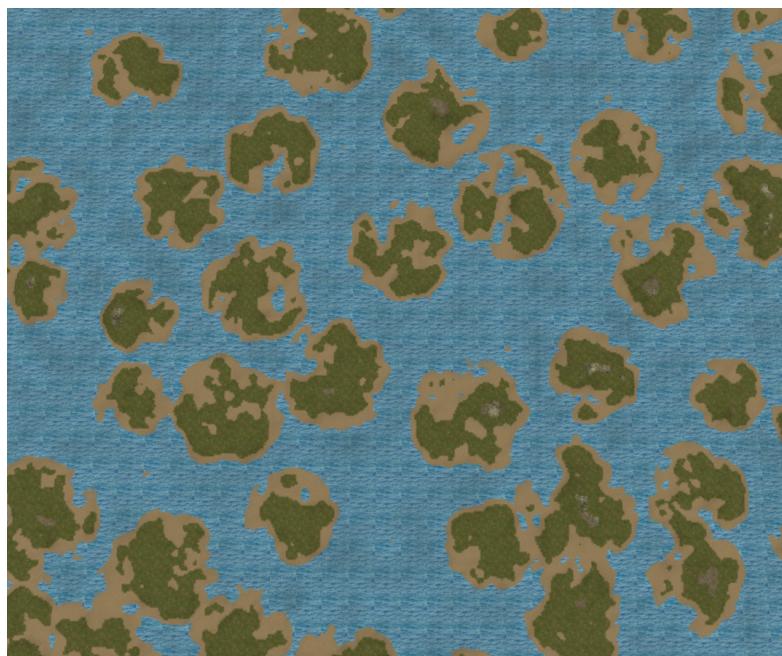


Abb. 3.5. Kombination mit einer Gewichtung die von einer Kurve vorgegeben wurde

3.2.2 Potenzieren

Das Potenzieren der *Noise-Map* kam von der Überlegung, dass ein Wert zwischen Null und Eins potenziert mit einem positiven Wert wieder einen Wert zwischen Null und Eins ergibt. Es ist also keine Gewichtung wie bei dem vorherigen Beispiel nötig.

Der Exponent kann wiederum über eine Kurve abhängig von der Distanz festgestellt werden. Werte über Eins drücken den Wert der *Noise-Map* dabei nach unten und Werte über Eins nach oben(siehe Abb. 3.6).

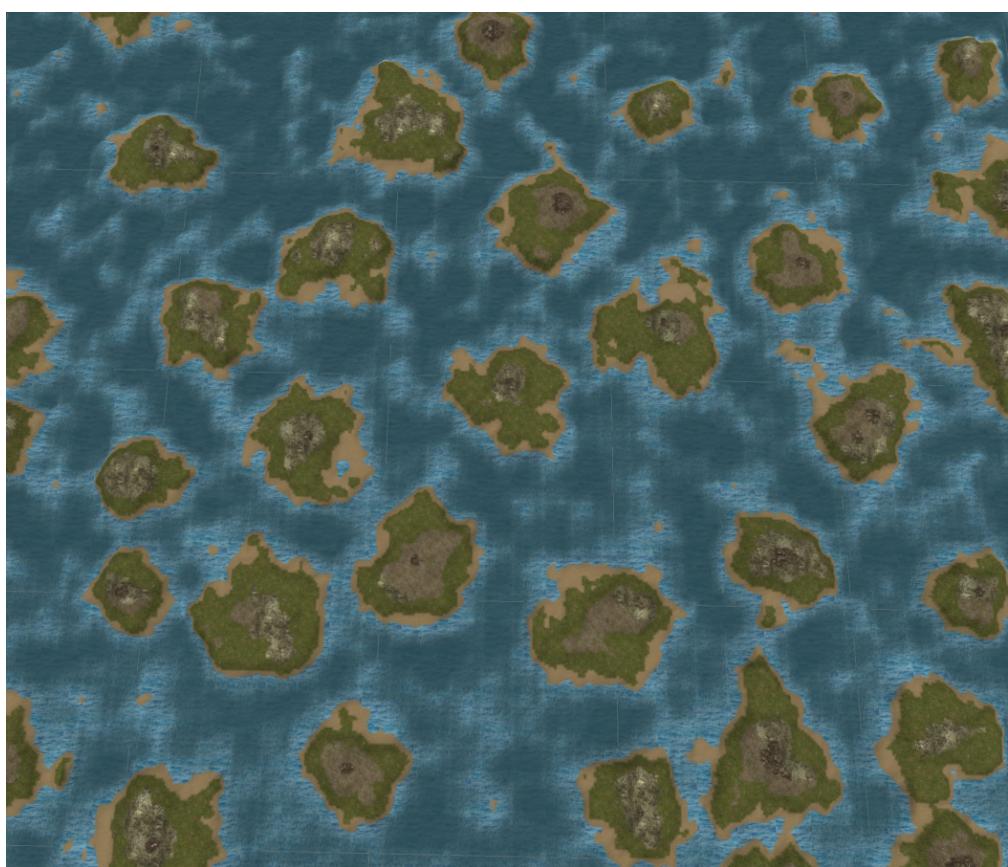


Abb. 3.6. Kombination mit einem Exponenten der über eine Kurve definiert wurde

4

Kartengenerierung

Die Kartengenerierung ist das verteilen der verschiedenen Gebiete, basierend auf der Höhenkarte und zusätzlichen *Noise-Maps*. Dies geschieht in der Klasse *SplatmapCreator.cs*. Dabei wird nicht nur entschieden welche Textur vergeben wird, es wird auch eine *Tile-Map* mit mehr Informationen abgespeichert.

Wenn das Programm gestartet wird kann man durch einfaches Anklicken des Terrains feststellen welches Gebiet an dem jeweiligen Punkt liegt. Die Informationen über jedes Gebiet bestehen im Moment lediglich aus einem Namen der in der Konsole ausgegeben wird. Dies kann jedoch einfach erweitert werden.

Zuerst wird die Karte in grundlegende Gebiete, sogenannte *Areas* aufgeteilt. Dies geschieht anhand des Höhenwertes, indem ein Wert angeben wird bis zu dem das Gebiet entstehen soll. Die *Areas* werden der Reihe nach abgearbeitet, was bedeutet dass falls ein fraglicher Punkt die Bedingung einer *Area* erfüllt, alle weiteren *Areas* nicht mehr beachtet werden.

Jede *Area* kann jedoch mehrere *Subareas* enthalten. Die Entscheidung hier funktioniert wie bei den *Areas*, nur dass hier nicht die Höhe sondern eine weitere *Noise-Map* beachtet wird. Diese *Noise-Map* muss vorher definiert werden und dann natürlich dort angegeben werden.

Hier zeigt sich auch ein wesentlicher Vorteil der Histogrammäqualisierung. Nehmen wir zum Beispiel an, man möchte etwa zehn Prozent des Wassers mit einer anderen Textur belegen. Man würde annehmen das dafür der Schnitt auf 0,1 festgelegt werden muss. Lässt man die generierte *Noise-Map* unverändert ist so gut wie nichts von dieser Textur zu sehen(siehe Abb. 4.1). Aktiviert man allerdings die Äqualisierung entspricht das Ergebnis schon eher den Erwartungen(siehe Abb. 4.2).

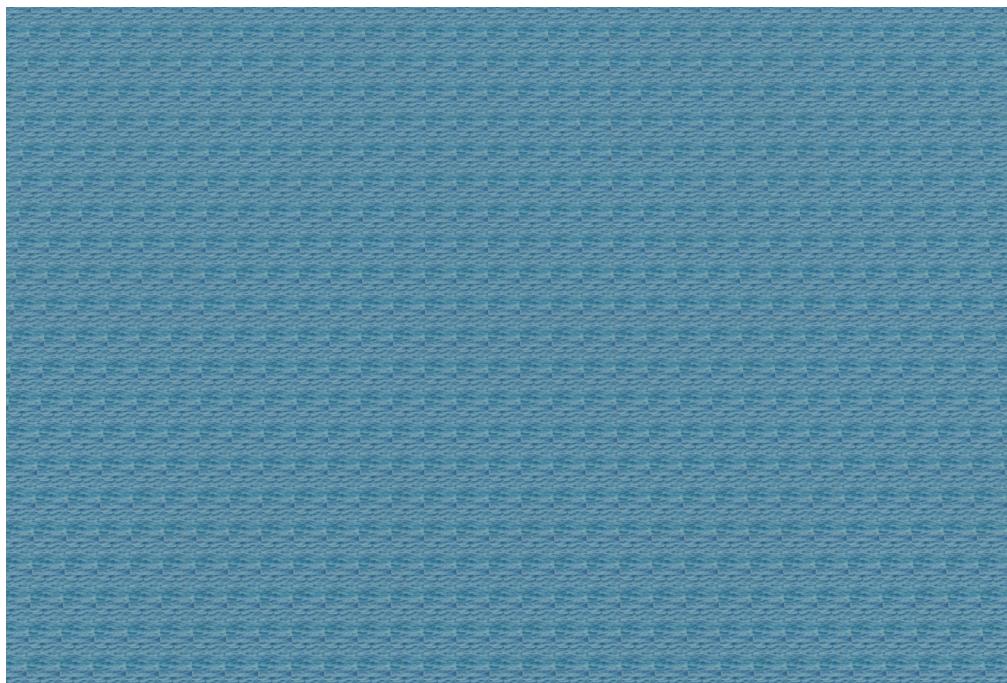


Abb. 4.1. Die Karte mit unveränderter *Noise-Map*

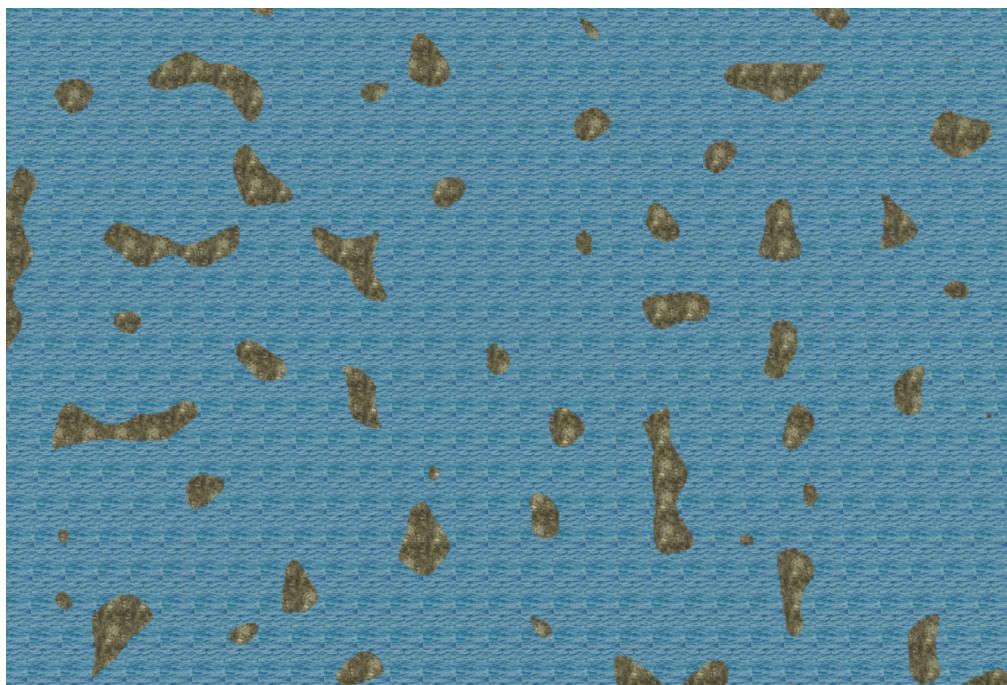


Abb. 4.2. Die Karte mit äqualisierter *Noise-Map*

5

Editor

Die Skripte die im Laufe dieser Arbeit entstanden sind arbeiten mit dem in *Unity* eingebauten Terrain. Das Terrain übernimmt dabei die grafische Darstellung und hat viele vorgegebene Einstellungsmöglichkeiten.

Ebenso haben die entstandenen Skripte viele Einstellungsmöglichkeiten, die hier erklärt und dokumentiert werden sollen. Für einige Skripte wurden Editorskripte erstellt um zum Beispiel Knöpfe anzeigen zu können. Die Einstellungen können über *Prefabs* einfach gespeichert und geladen werden.

5.1 Controller

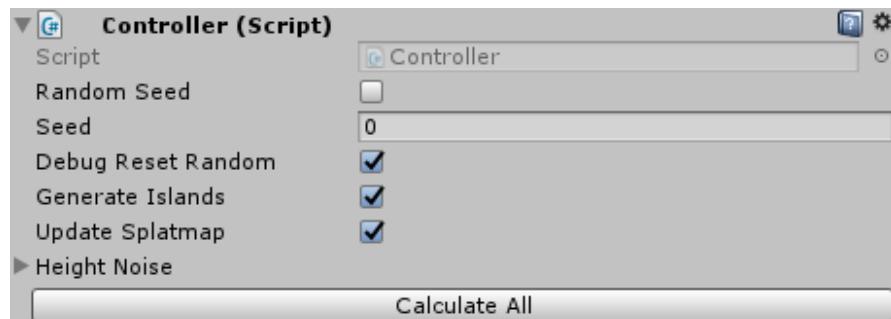


Abb. 5.1. Der Editor für den Controller

- **Random Seed:** Dies gibt an ob der Seed für die Zufallszahlengenerierung auf der Zeit basieren soll oder manuell angegeben wird.
- **Seed:** Ein ganzzahliger Wert für die initialisierung des Zufallszahlgenerators.
- **Debug Reset Random:** Dies gibt an ob der Zufallszahlengenerator vor jeder Nutzung zurückgesetzt werden soll. Das kann Vorteile haben wenn man zum Beispiel den *Splatmap Generator* neu einstellt und nicht bei jedem Versuch verschiedene Ergebnisse haben möchte.
- **Generate Islands:** Dies stellt die Benutzung der Inselmittelpunkte ein und aus und entscheidet damit darüber ob die Skripte *Island Creator* und *Island Distribution* einen Einfluss haben.

- **Update Splatmap:** Dies gibt an ob der *Splatmap Creator* neu berechnet werden soll.
- **Height Noise:** Dies stellt die grundlegende *Noise* für die Höhenberechnung dar. Die einzelnen Einstellungen werden im Abschnitt Noise Creator erklärt.
- **Calculate All:** Dies startet die Berechnung mit allen angegebenen Einstellungen.

5.2 Island Creator

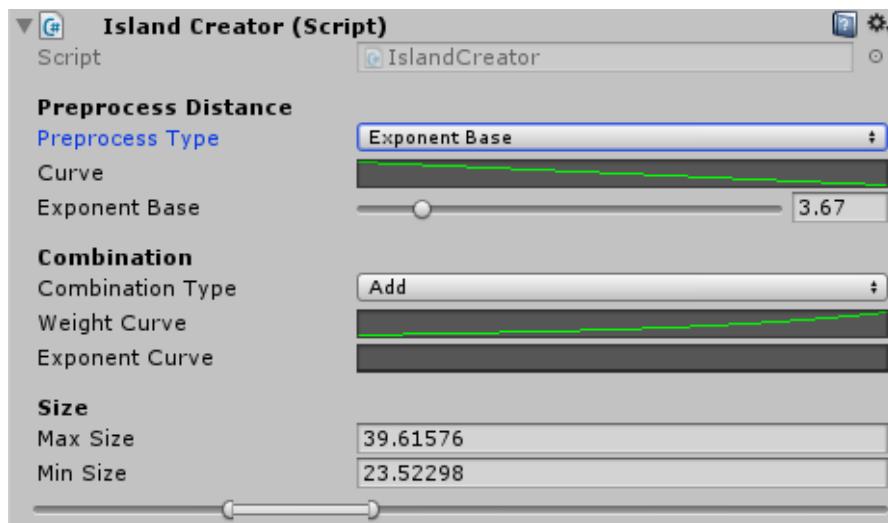


Abb. 5.2. Der Editor der das Aussehen der Inseln verändert

- **Preprocess Type:** Dies gibt an welche der beiden nächsten Variablen verwendet wird um die Distanz für die Kombination vorzubereiten.
- **Curve:** Mit dieser Kurve wird die Distanz verändert. Hier ist zu beachten, dass die Distanz zwischen Null und Eins liegt. Ebenso müssen die Ergebnisse zwischen Null und Eins liegen.
- **Exponent Base:** Dies gibt die Basis für das Potenzieren an. Somit lässt sich schnell eine stetig fallende Kurve definieren(siehe Abb. 3.2).
- **Combination Type:** Dies gibt die Art der Kombination an.
- **Weight Curve:** Wenn die Methode *Add* ausgewählt wurde, wird hiermit die Gewichtung anhand der Distanz festgelegt. Das Ergebnis muss wiederum zwischen Null und Eins liegen.
- **Exponent Curve:** Wenn die Methode *Exponent* ausgewählt wurde, wird hiermit der Exponent anhand der Distanz festgelegt. Außerdem sind dann alle Einstellungen unter *Preprocess* irrelevant. Hier sollen auch Ergebnisse größer Eins heraus kommen, da diese die Höhenwerte nach unten drücken.
- **Max Size: Min Size:** Diese beiden Werte geben die Größe der Inseln an. Jede Insel ermittelt einen zufälligen Wert zwischen diesen beiden Werten. Sie können auch über den Regler eingestellt werden.

5.3 Island Distribution

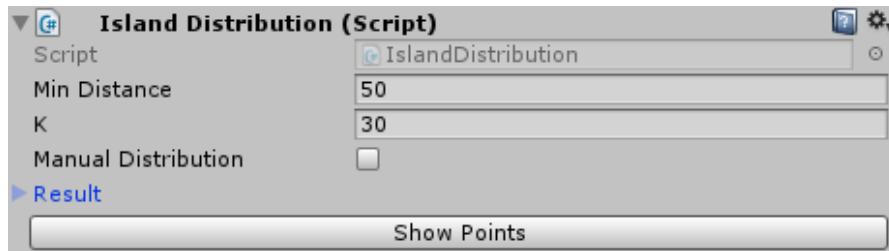


Abb. 5.3. Der Editor für die Inselverteilung

- **Min Distance:** Dies gibt den Mindestabstand zwischen zwei Punkten an.
- **K:** Hiermit wird angegeben wie oft von einem Punkt aus versucht wird einen Weiteren zu finden. Je höher dieser Wert ist, desto besser aufgefüllt wird das Terrain werden, allerdings steigt dadurch auch die Rechenzeit.
- **Manual Distribution:** Dies gibt an ob die Punkte manuell angegeben werden. Dadurch werden die beiden oberen Einstellungen obsolet.
- **Result:** Hier können die errechneten Punkte eingesehen werden oder selbst definiert werden.
- **Show Points:** Mit diesem Knopf werden die Punkte berechnet und in Form von „Stacheln“ auf der Höhenkarte angezeigt.

5.4 Splatmap Creator

- **Noises:** Hier können beliebig viele *Noise-Maps* definiert werden. Der Name ist hier wichtig, da mit diesem in den Subareas angegeben wird welche *Noise* verwendet wird.
- **Areas:** Areas bestehen aus einem Namen einer Textur und dem Wert *Cut*. Dieser gibt an bis zu welcher Höhe das entsprechende Areal angewendet wird.
- **Subareas:** Jede Area kann eine beliebige Anzahl von Subareas haben. Diese funktionieren ähnlich wie Areas, jedoch entscheiden sie anhand der angegebenen *Noise-Map*.
- **Water Level:** Zuletzt kann noch die Höhe des Wassers angegeben werden. Dieses wird automatisch skaliert und positioniert, falls sich die Größe des Terrains verändert hat.
- **Create Splatmap:** Hierdurch wird nur der *Splatmap Creator* neu berechnet. Die Höhenkarte bleibt dabei gleich.

5.5 Map Analyser

Der *Map Analyser* ist nur eine Hilfestellung beim Einstellen des Editors. Hier wird die Anzahl der Werte, das Minimum, das Maximum und ein Histogramm angezeigt. Die Genauigkeit des Histogramms kann über den Regler angegeben werden.

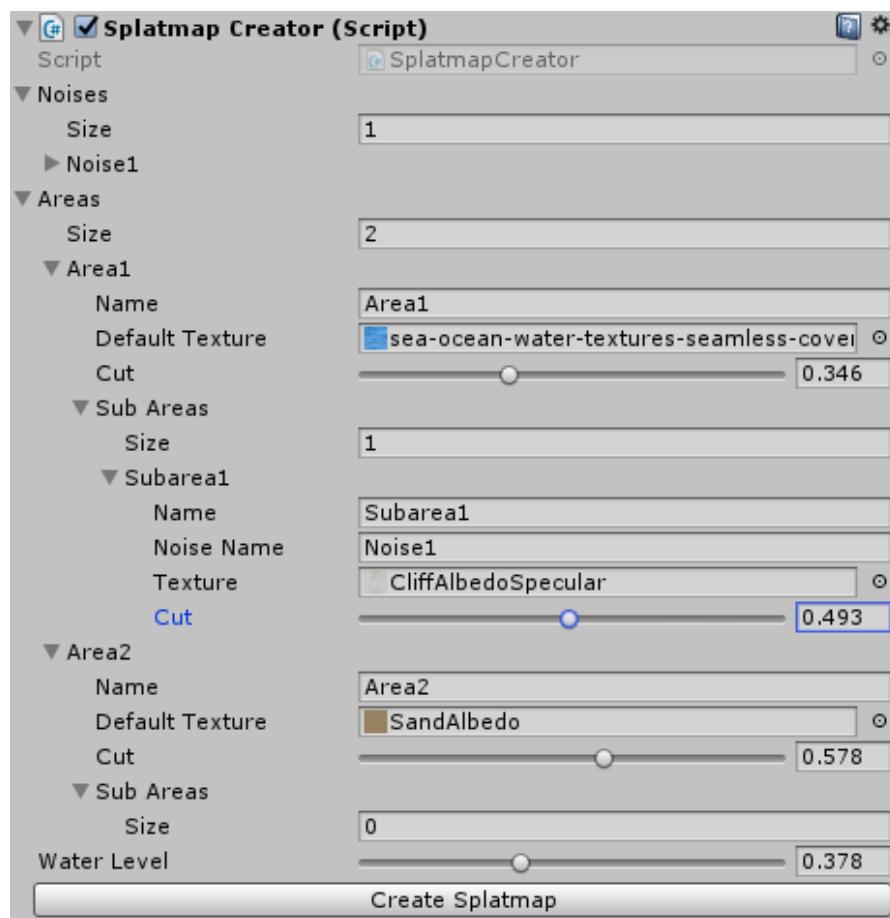


Abb. 5.4. Der Editor für das erstellen der verschiedenen Gebiete

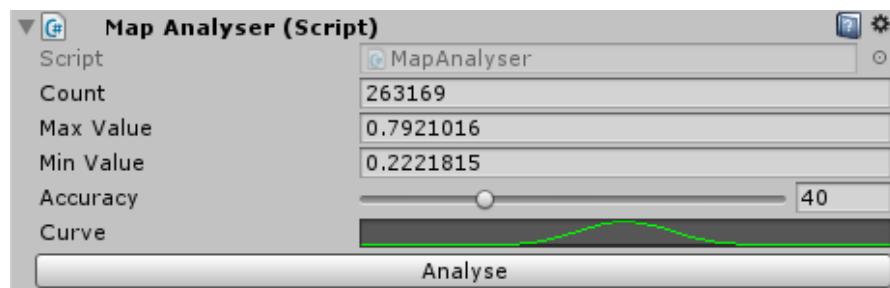


Abb. 5.5. Der Editor für das analysieren der Höhenkarte

5.6 Noise Creator

- **Name:** Der Name der *Noise* ist nur im *Splatmap Creator* wichtig, da hiermit von den *Subareas* auf die *Noise* zugegriffen wird.
- **Random:** Dies gibt an ob die beiden folgenden Variablen zufällig ausgewählt werden. Für Testzwecken kann es nützlich sein die *Noise-Map* nur um wenige Punkte zu verschieben.
- **X Offset: Y Offset:** Diese Variablen geben an, an welchem Punkt des theoretisch unendlichen Feldes die *Noise-Map* starten soll.

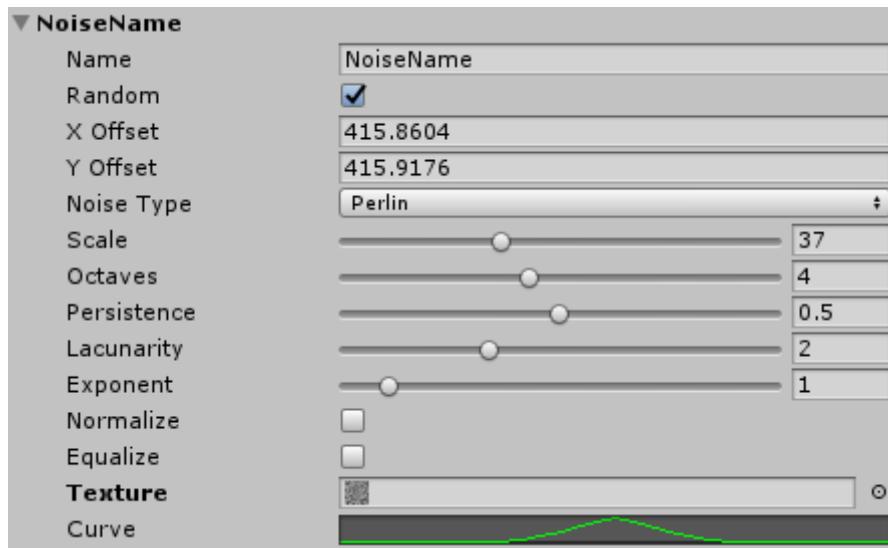


Abb. 5.6. Der Editor für das definieren einer *Noise*

- **Noise Type:** Hier wird angegeben mit welchem Algorithmus die *Noise* berechnet werden soll. Bisher gibt es nur eine selbst implementierte *Perlin-Noise* und eine von *Unity* vorgegebene. Der von *Unity* vorgegebene *Noise*-Algorithmus ist jedoch nicht zuverlässig, da hier auch Wert über Eins heraus kommen können[Uni].
- **Scale:** Die Skalierung gibt die Frequenz der ersten Oktave an. Die Frequenz zu erhöhen entspricht einem Heraus-zoomen aus der *Noise-Map*.
- **Octaves:** Dies gibt die Anzahl der Oktaven an, wobei mit jeder Oktave die Frequenz steigt aber die Gewichtung sinkt. Diese Oktaven werden dann zusammengelegt um ein detaillierteres Bild zu erzeugen [Fra].
- **Persistence:** Die Persistenz gibt an wie stark die Gewichtung mit jeder Oktave sinkt. Ein guter Richtwert hier ist 0,5, sodass die Gewichtung für jede Oktave halbiert wird.
- **Lacunarity:** Dieser Wert gibt an wie stark die Frequenz mit jeder Oktave steigt.
- **Exponent:** Das Ergebnis der *Noise-Map* wird mit diesem Wert potenziert. Bei einem Wert über Eins drückt dies den unteren Bereich weiter nach Unten. Dies kann praktisch sein um Berge besser zu betonen.
- **Normalize:** Die Normalisierung versucht den Wertebereich der *Noise* zu verbreitern, da sich sonst der Großteil der Werte im mittleren Bereich befindet. Dies funktioniert jedoch nicht so gut wie mit der Äqualisation.
- **Equalize:** Die Äqualisation verwendet ein kumulatives Histogramm um den Wertebereich zu verbreitern. Dies ist besonders beim erstellen der *Subareas* sehr nützlich(siehe Kapitel 4).
- **Texture:** Hier wird ein einfaches Graustufenbild mit den errechneten Werten angezeigt.
- **Curve:** Die Kurve zeigt ein Histogramm, also die Verteilung der Werte über den Wertebereich, an.

6

Zusammenfassung und Ausblick

Die meiste Zeit der Entwicklung wurde für Kombination aus *Noise-Map* und den vorgegebenen Punkten gebraucht. Dies war allerdings zu erwarten, da es in diesem Bereich noch keine bekannte Vorarbeit gab. Das Ergebnis ist durchaus zufriedenstellend und gibt dem Benutzer des Editors viele Einstellungsmöglichkeiten. Zum Beispiel können statt Inseln auch Berge oder Täler verteilt werden.

Die Form der Inseln könnte jedoch besser variiert werden, wenn die Größe in jede Richtung unterschiedlich ist. Hierzu müssten zuerst geeignete Verfahren gefunden werden.

Erweiterungsmöglichkeiten gibt es auch beim Verteilen der Punkte und beim Erstellen der *Noise-Map*. Wie bereits erwähnt gibt es viele andere *Noise*-Verfahren, wie *Simplex-Noise*, *Flow-Noise* und *Curl-Noise*, die hier eingebaut werden könnten [Proa].

Interessant sind auch die Möglichkeiten für das Verteilen der Punkte. Zum Beispiel können die Mindestabstände zwischen den Punkten durch eine *Noise-Map* beeinflusst werden(siehe Abb. 6.1).

Weiterhin könnte man Inselgruppen erstellen, indem man eine *Poisson-Disk*-Verteilung benutzt um Mittelpunkte für Inselgruppen zu verteilen. In der Nähe von diesen würde man dann im nächsten Schritt die Inselmittelpunkt verteilen.

Es können auch ganz andere Verfahren, wie zum Beispiel ein Masse-Feder-System verwendet werden um die Punkte zu verteilen[Mas].

Eine weitere Verbesserung betrifft die grafische Darstellung des Terrains. Zum Beispiel kann eine entstandene Graslandschaft hügeliger sein als dies gewollt ist, da hier Gebäude gebaut werden sollen(siehe Abb. 6.2).

Um dies zu lösen könnte die Höhe nachbearbeitet und bestimmte Wertebereiche zusammengeschoben werden um eine flachere Landschaft zu erzeugen(siehe Abb. 6.3).

Die anfangs angesprochene unendliche Landschaft ist im vorliegenden Projekt nicht vorhanden. Die Karte kann zwar sehr groß berechnet werden und damit eine scheinbare Unendlichkeit bieten. Besser wäre es allerdings die Karte dort weiter zu generieren, wo sich der Spieler ausbreitet.

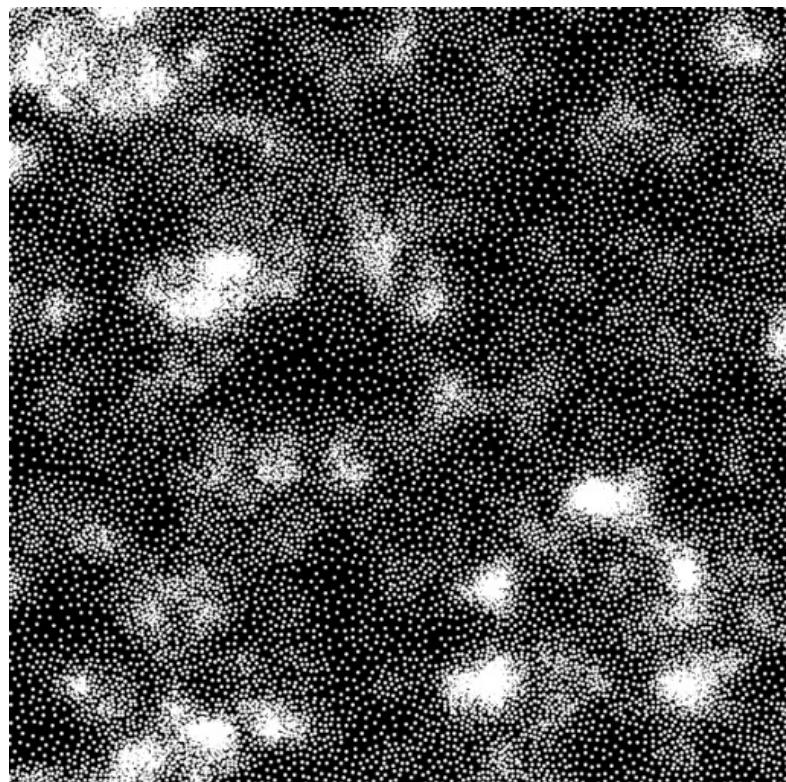


Abb. 6.1. *Poisson-Disk-Sampling beeinflusst durch eine Noise-Map*



Abb. 6.2. Die entstandene Graslandschaft entspricht zwar der erwünschten Größe, ist jedoch zu Hügelig um Gebäude darauf zu platzieren



Abb. 6.3. In *Anno 1404* sind die Inseln größtenteils flach

Literaturverzeichnis

- Aeq. *Wikipedia - Histogrammäqualisation.*
[https://de.wikipedia.org/wiki/Punktoperator_\(Bildverarbeitung\)
#Histogramm.C3.A4qualisation.](https://de.wikipedia.org/wiki/Punktoperator_(Bildverarbeitung)#Histogramm.C3.A4qualisation)
- Bac. *Bachelorarbeit Projekt.*
[https://github.com/hannes1991113/ProMapGen.](https://github.com/hannes1991113/ProMapGen)
- Fra. *Fractional Brownian Motion.*
[https://code.google.com/archive/p/fractalterraingeneration/
wikis/Fractional_Brownian_Motion.wiki.](https://code.google.com/archive/p/fractalterraingeneration/wikis/Fractional_Brownian_Motion.wiki)
- Isl. *Islands from Noise.*
[http://www.redblobgames.com/maps/terrain-from-noise/#islands.](http://www.redblobgames.com/maps/terrain-from-noise/#islands)
- Mas. *Wikipedia - Masse-Feder-System.*
[https://de.wikipedia.org/wiki/Masse-Feder-System_
\(Computergrafik\).](https://de.wikipedia.org/wiki/Masse-Feder-System_(Computergrafik))
- Nor. *Wikipedia - Histogrammnormalisierung.*
[https://en.wikipedia.org/wiki/Normalization_\(image_
processing\).](https://en.wikipedia.org/wiki/Normalization_(image_processing))
- Pera. *Perlin Noise.*
[http://mrl.nyu.edu/~perlin/paper445.pdf.](http://mrl.nyu.edu/~perlin/paper445.pdf)
- Perb. *Perlin Noise Implementierung.*
[https://gist.github.com/Flafla2/f0260a861be0ebdeef76.](https://gist.github.com/Flafla2/f0260a861be0ebdeef76)
- Poia. *Poisson Disk Distribution.*
[http://devmag.org.za/2009/05/03/poisson-disk-sampling/.](http://devmag.org.za/2009/05/03/poisson-disk-sampling/)
- Poib. *Poisson Disk Implementierung.*
[https://github.com/RazorYhang/PoissonDiskGeneratorForUnity.](https://github.com/RazorYhang/PoissonDiskGeneratorForUnity)
- Proa. *Procedural Noise Functions.*
[https://www-sop.inria.fr/reves/Basilic/2010/LLCDDELPZ10/
LLCDDELPZ10STARPNF.pdf.](https://www-sop.inria.fr/reves/Basilic/2010/LLCDDELPZ10/LLCDDELPZ10STARPNF.pdf)
- Prob. *Wikipedia - Prozedurale Generierung.*
[https://de.wikipedia.org/wiki/Prozedurale_Synthese.](https://de.wikipedia.org/wiki/Prozedurale_Synthese)
- Uni. *Unity Perlin Noise.*
[https://docs.unity3d.com/ScriptReference/Mathf.PerlinNoise.
html.](https://docs.unity3d.com/ScriptReference/Mathf.PerlinNoise.html)

A

Erklärung der Kandidatin / des Kandidaten

- Die Arbeit habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen- und Hilfsmittel verwendet.

Datum

Unterschrift der Kandidatin / des Kandidaten