

## Maulwurfshügel in QGIS

### Reliefdarstellung in Maulwurfshügelmanier auf der Grundlage digitaler Höhendaten



„Maulwurfshügel“ sind eine altertümliche Art der Reliefdarstellung, die verbreitet in Karten des 16. bis 18. Jahrhunderts begegnet sowie, inspiriert hiervon, in Karten des Fantasy-Genres. Maulwurfshügel bieten sich an, wo eine bestimmte antike Ästhetik gewünscht ist, während geografische Genauigkeit nicht im Vordergrund steht. Dieser Abriss erläutert, wie sich als SVG-Grafiken vorbereitete Hügel anhand heruntergeladener Höhendaten mit etwas zusätzlicher Arbeit in QGIS automatisch auf der Karte anordnen lassen.<sup>1</sup>

Das geschieht im Wesentlichen in sechs Schritten:

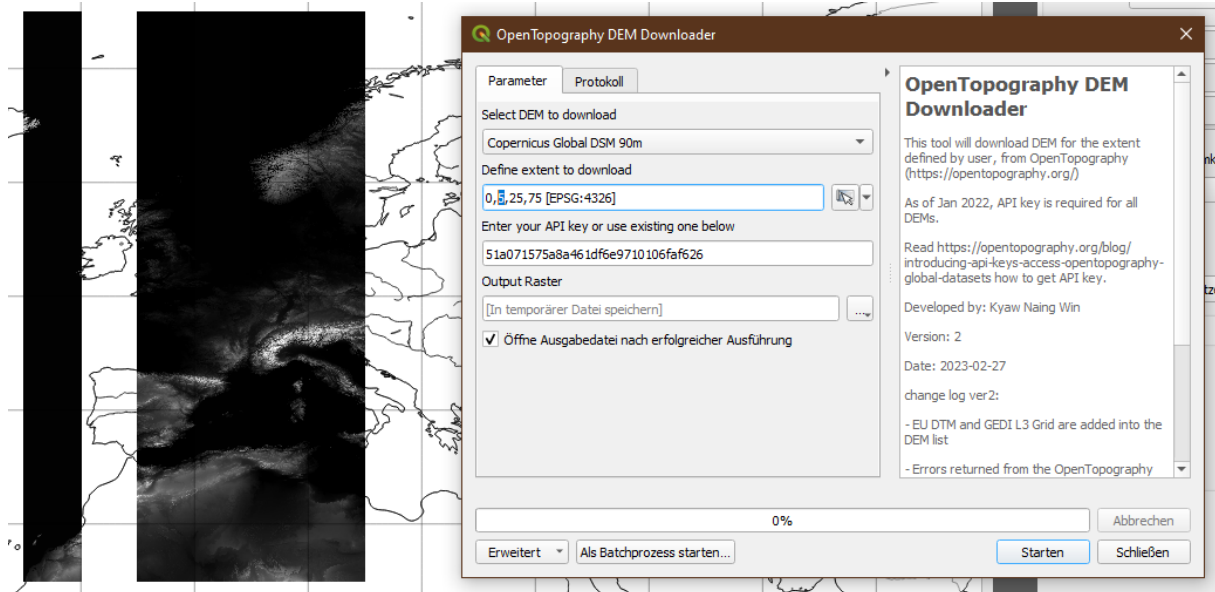
- I. Herunterladen der Höhendaten,
- II. Umwandeln der Höhenkarte in eine Steilheitskarte,
- III. Ableiten von Mittel- und Hochgebirgsflächen aus der Steilheitskarte,
- IV. Anlegen von Quellebenen und Zielebene für die Hügelpunkte,
- V. Generieren der Punkte für verschiedene Maßstäbe und
- VI. Darstellung der Punkte mit Hügelgrafiken.

---

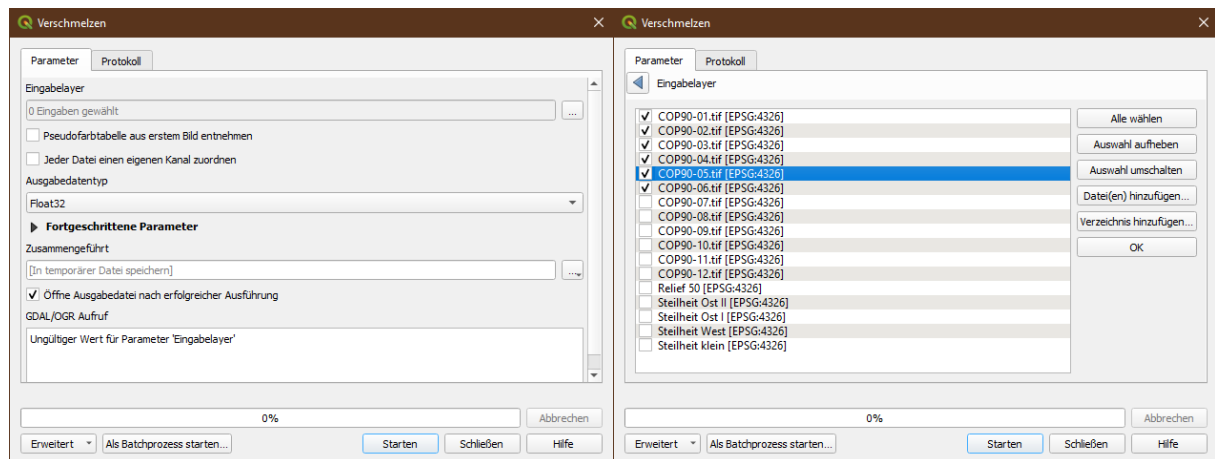
<sup>1</sup> Getestet wurde es hier in QGIS 3.28 und 3.34. — Die hier verwendeten Python-Skripte habe ich nicht allein programmiert, da ich darin nicht bewandert bin. ChatGPT war mir eine große Hilfe!

## I. Herunterladen der Höhendaten

Zunächst wird eine Höhenkarte für das betreffende Gebiet benötigt. Dafür können wir das Plugin *OpenTopography DEM Downloader* verwenden und z.B. Copernicus-90-Daten herunterladen. Da der Server stets nur begrenzte Volumina herausrückt, müssen sie ggf. gestückelt heruntergeladen werden; im nachstehenden Beispiel geschieht dies in 5° breiten Streifen:



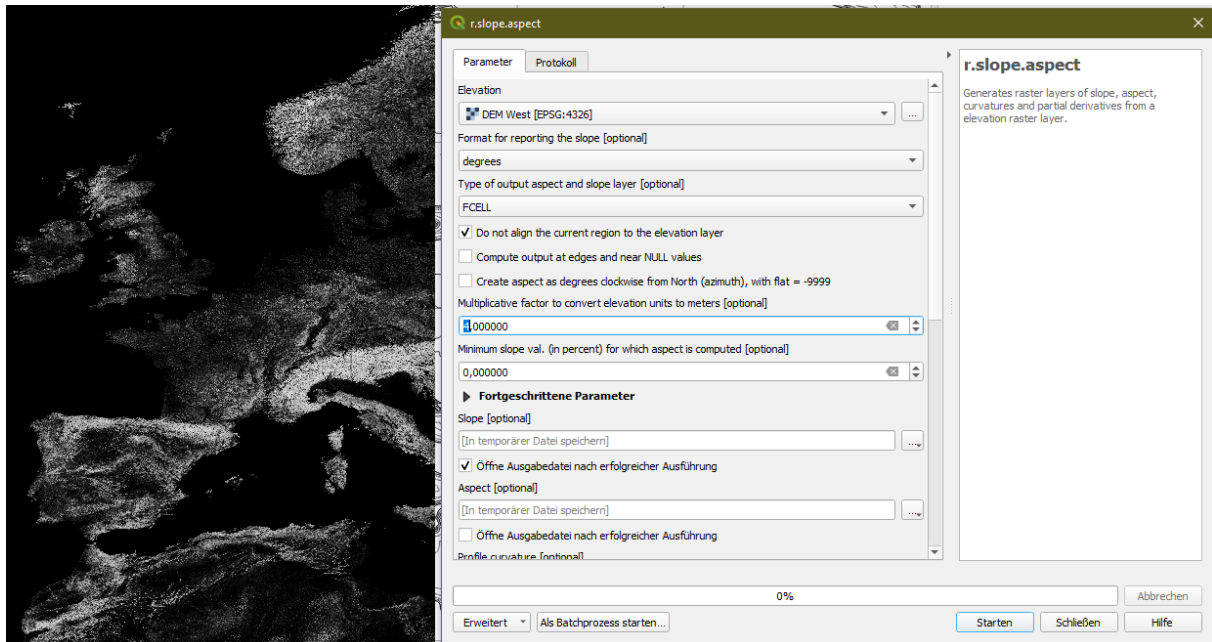
Die vielen einzelnen Ebenen, die dabei entstehen, sollten vor ihrer weiteren Verarbeitung zu einer Ebene vereinigt werden. Wenn QGIS dabei wegen der hohen Datenmengen an seine Leistungsgrenzen stößt, muss man sich ggf. mit mehreren, möglichst wenigen Ebenen begnügen. Die Funktion zum Verschmelzen findet sich im Menü unter *Raster* → *Sonstiges* → *Verschmelzen*:



## II. Umwandeln der Höhenkarte in eine Steilkarte

Nun haben wir eine schöne Höhenkarte, die aber noch nicht viel nützt – denn wir wollen ja keine Ebene mit Hügeln besäen, nur weil es eine Hochebene ist, während wir umgekehrt Küstengebirge auch dann abbilden wollen, wenn sie nicht weit über Null hinausragen. Was wir stattdessen brauchen, ist also eine Karte, die nicht die Höhe darstellt, sondern die Steilheit des Geländes.

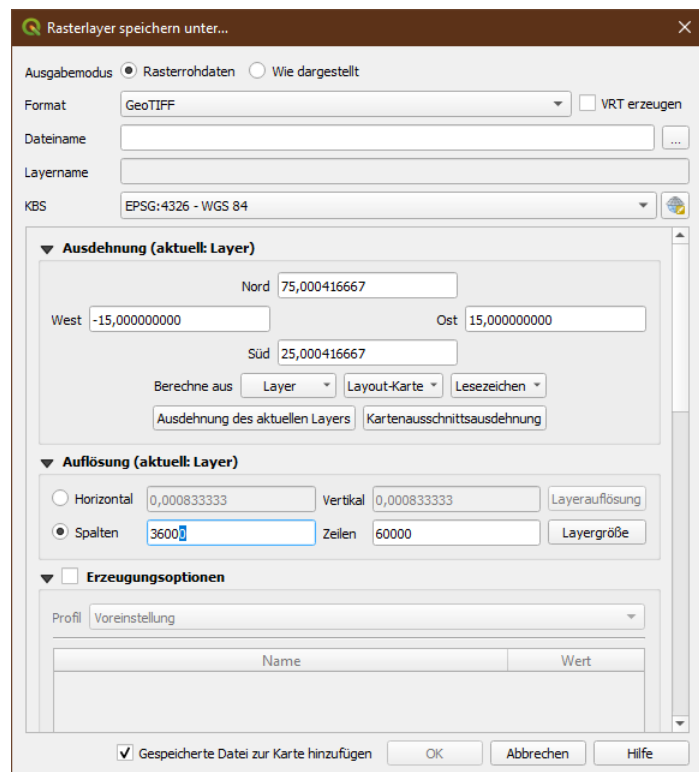
Dafür bemühen wir die GRASS-GIS-Funktion *r.slope.aspect*. In deren Fenster muss unter *Fortgeschrittene Parameter* nur das Häkchen hinter ‚Slope‘ bleiben, alle weiteren können weg. Als Multiplikator habe ich 4 gewählt; gewiss gehen auch andere Werte.



Auf diese Weise entsteht eine Karte, die nicht mehr zeigt, wie hoch gelegen, sondern wie steil zerklüftet das Gelände ist. Man beachte etwa die Hochebenen Spaniens und Marokkos: Sie sind nicht mehr grau (weil hoch), sondern schwarz (weil eben).

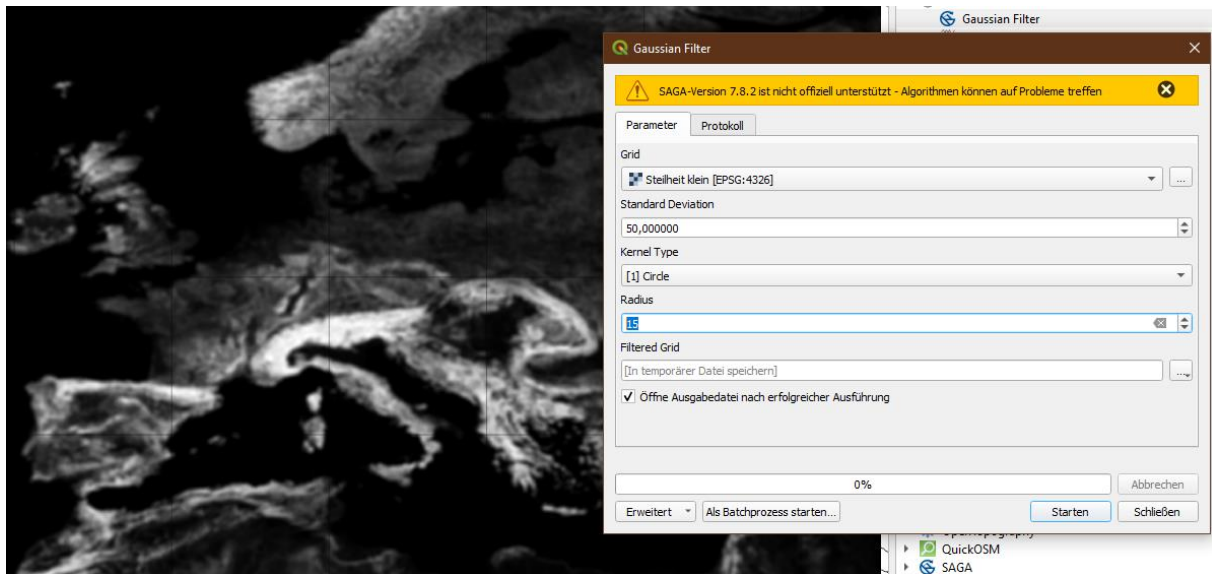
Um die weitere Verarbeitung zu erleichtern, verkleinern wir die Grafik erst einmal auf ein Zwanzigstel, indem wir nach Rechtsklick auf die Ebene —> *Export* —> *Speichern als ...* wählen. Im nun geöffneten Fenster entfernen wir bei Spalten und Zeilen je eine Null und teilen den Rest durch zwei, wählen unter *Dateiname* einen Speicherort und speichern die Grafik als neue Ebene (s. rechts).

Diese verkleinerte Karte ist allerdings zur weiteren Verwendung, d.h. Vektorisierung, noch immer zu feinkörnig. Wir belegen sie darum mit einer Gauß'schen Unschärfe, indem wir aus den SAGA-Werkzeugen den *Gaussian Filter* anwenden<sup>2</sup>; im hiesigen Beispiel schien dabei ein Radius von 15 angemessen (s. folgende Seite).



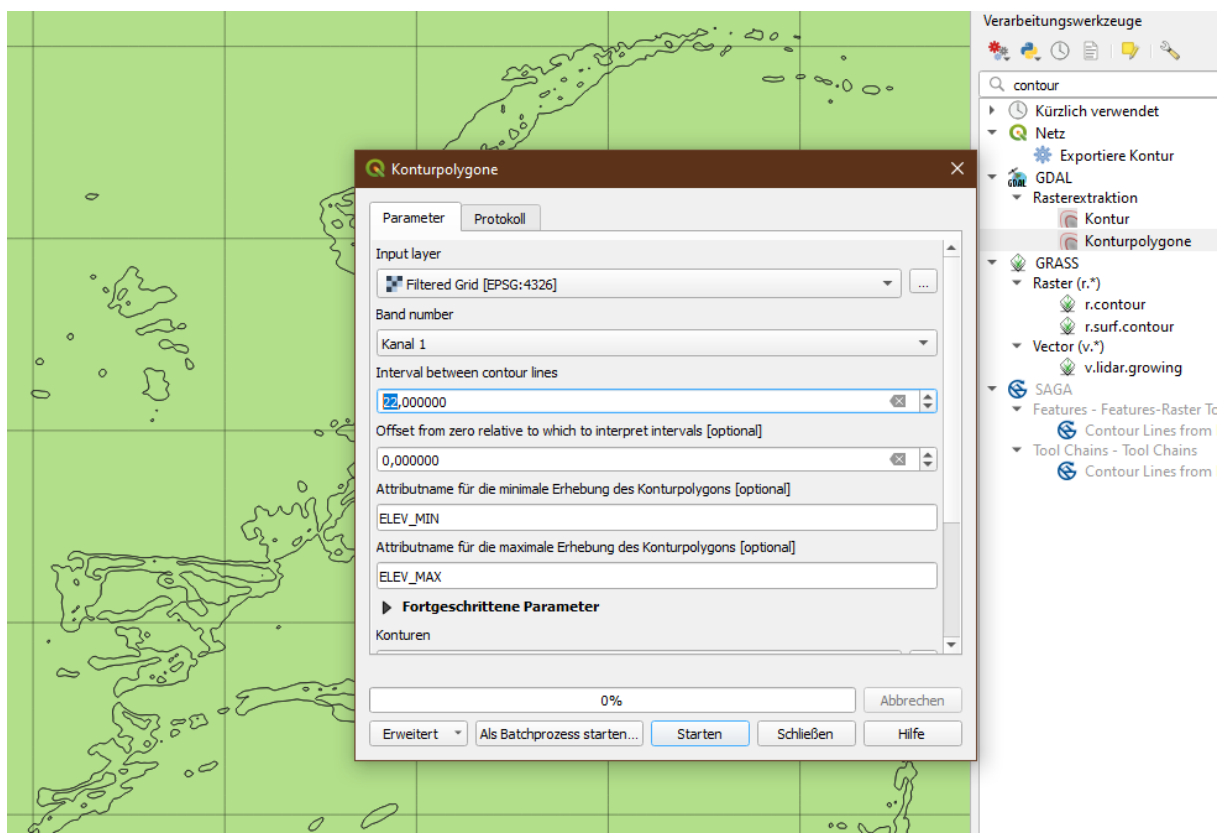
<sup>2</sup> QGIS 3.28 scheint die letzte Version zu sein, die SAGA-Werkzeuge noch von Haus aus unterstützt. In QGIS 3.34 ist dafür schon das Plugin „Processing Saga NextGen Provider“ und seine Einstellung unter *Einstellungen* —> *Optionen* —> *Verarbeitung* —> *Datenanbieter* notwendig.





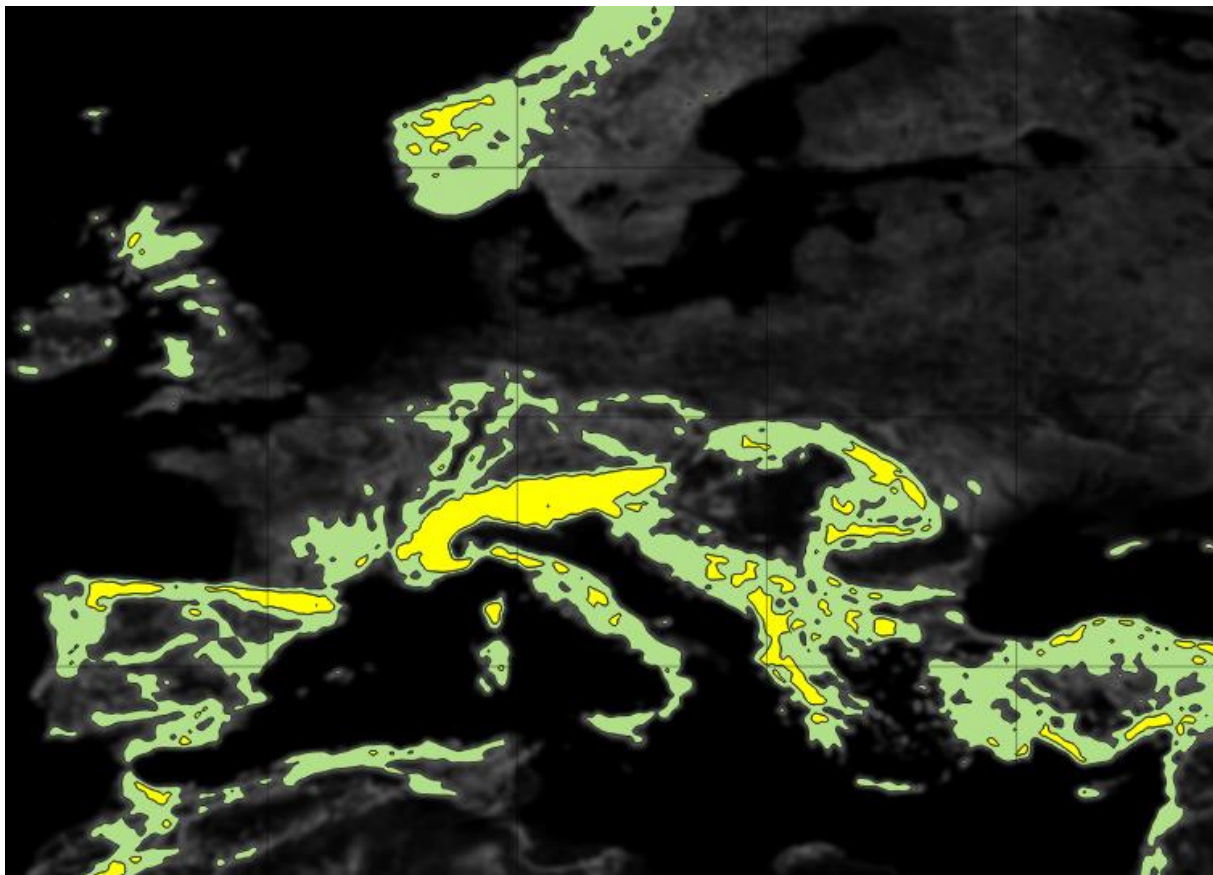
### III. Ableiten von Mittel- und Hochgebirgsflächen aus der Steilheitskarte

Nun müssen die von Mittel- und Hochgebirge bedeckten Flächen gefunden und als Polygone extrahiert werden. Dafür nutzen wir die GDAL-Funktion *Konturpolygone*, die eigentlich für Höhenlinien gedacht ist. Im vorliegenden Beispiel führt ein Intervall-Wert von 22 zu passenden Ergebnissen:



Auf der neu entstandenen Ebene haben wir jetzt drei Polygone: Eines erstreckt sich bis zu den Kartenrändern und über alle relativ ebenen Flächen einschließlich der Meere; dieses können wir löschen. Übrig bleiben die zwei Polygone für Mittelgebirgs- und Hochgebirgsflächen. Sollte es noch ein weiteres für die steilsten Gipfel geben, können wir es ebenfalls löschen, falls es

vernachlässigbar klein ist, oder wir starten die Prozedur mit einem höheren Wert als 22 erneut. (Gibt es gar noch mehr Polygone, war der Wert ohnehin zu niedrig.)



#### IV. Anlegen von Quellebenen und Zielebene für die Hügelpunkte

Die beiden verbleibenden Polygone verteilen wir auf zwei Ebenen, die wir „Mittelgebirge“ und „Hochgebirge“ nennen. Zusätzlich erstellen wir eine weitere Vektorebene, deren Geometrie in Punkten besteht und die wir „Gebirgspunkte“ nennen. (Diese Ebenen kann man natürlich auch anders nennen, nur muss man dann die Zufallspunkte-Funktionen, die sogleich erläutert werden, an den entsprechenden Stellen ebenfalls ändern, s. Anhang.) Der Ebene „Gebirgspunkte“ müssen – entweder gleich bei ihrer Erstellung oder später in den Layer-Eigenschaften – die folgenden Felder hinzugefügt werden (außer „id“, welches schon besteht, aber nutzlos ist):

Layerereigenschaften - Gebirgspunkte — Felder							
	Id	Name	Alias	Typ	Typname	Länge	Genauigkeit
	123 0	id		Ganzzahl (Integer 64 bit)	Integer64	10	0
	abc 1	SVG		Text (string)	String	10	0
	123 2	Massstab		Ganzzahl (Integer 64 bit)	Integer64	10	0
	abc 3	Kategorie		Text (string)	String	10	0
	123 4	Versatz		Ganzzahl (Integer 64 bit)	Integer64	10	0
	1.2 5	Skalierung		Dezimal (double)	Real	10	3

Im Folgenden wird es darum gehen, die Flächen der zwei Polygone mit zufällig angeordneten Punkten zu füllen, die wiederum von zufällig gewählten SVG-Grafiken dargestellt werden. Die beiden erstgenannten Ebenen dienen dabei als Quell-Ebenen, aus denen heraus die Punkte generiert werden, und die dritte ist die Ziel-Ebene, in der sie abgespeichert werden.

Dies geschieht mittels einer Python-Funktion<sup>3</sup>, welche in zwei Varianten vorliegt – einmal für die Mittel- und einmal für die Hochgebirge (s. Anhang) – und die im Wesentlichen Folgendes tut:

1. über die Gesamtausdehnung des Polygons ein Gitter spannen,
2. in alle Kästchen des Gitters je einen Punkt zufällig platzieren,
3. alle Punkte wieder löschen, die dabei nicht innerhalb des Polygons landen,
4. alle Punkte wieder löschen, die zu nah an der Geometrie bestimmter anderer Ebenen liegen, von der sie sich fernhalten sollen (z.B. Flüsse, Küstenlinien, ...),
5. allen Punkten, die auch dann noch zu nah rechts oder links an jener Geometrie liegen, einen Attributwert von +1 oder -1 zuweisen, mit dem sie später davon abgerückt werden können (zu Sinn und Unsinn dieser Doppelung sogleich),
6. jedem Punkt zufällig einen Dateinamen von „MH01.svg“ bis „MH63.svg“ zuweisen (das sind die 63 Maulwurfshügel-Grafiken aus dem beiliegenden, vorbereiteten Satz) und
7. jedem Punkt einen zufälligen Skalierungs-Faktor von 0,9 bis 1,1 zuweisen, um zusätzlich etwas Abwechslung ins Kartenbild zu bekommen.

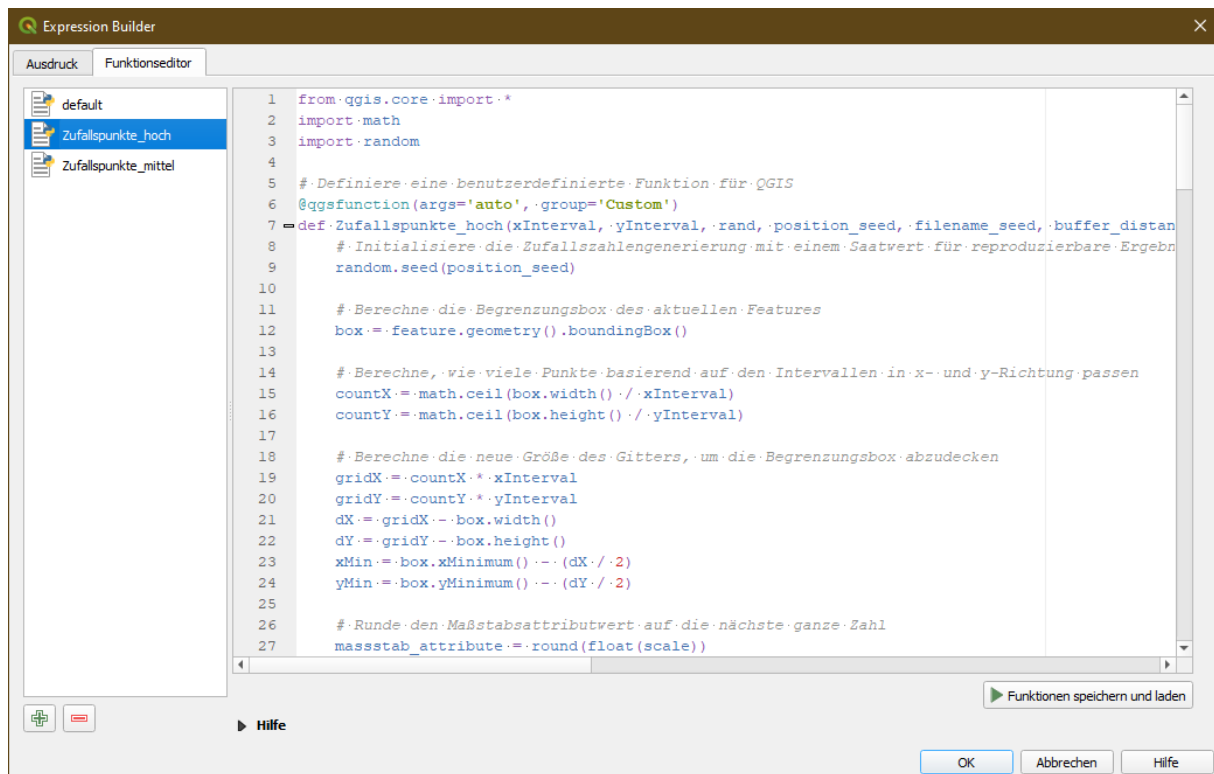
Punkte 4 und 5 verfolgen beide den Zweck, die Hügel nicht zu nah an die Geometrie bestimmter anderer Ebenen heranrücken zu lassen (die dafür namentlich in die Funktion eingetragen werden müssen, s. Anhang). In Punkt 4 wird um jene Geometrie ein Puffer gezogen, um sodann alle Punkte zu löschen, die darinnen liegen. Da unsere Hügel mehr breit als hoch sind, wäre es eigentlich schön, einen Puffer zu haben, der ebenfalls mehr breit als hoch ist. Das scheint in QGIS aber nicht vorgesehen zu sein. Mehr noch: Wenn, wie hier, Grad als Karteneinheiten dienen, sind diese in europäischen Breiten ausgerechnet mehr hoch als breit, und so gerät der Puffer genau umgekehrt, als er eigentlich sollte. Mir fiel hierfür keine andere Lösung ein, als in einem weiteren Schritt 5 noch einmal zu prüfen, welche Punkte denn dann noch im Puffer lägen, wenn sie etwas weiter rechts oder links lägen (und zwar um eine halbe Pufferbreite). Für diese Punkte wird dann ein positiver oder negativer Attributwert eingetragen, der später als Anknüpfungspunkt für einen seitlichen Versatz des Markers dienen kann<sup>4</sup>. (Einen Vorteil hat diese Methode: Punkte, die andernfalls gelöscht würden, bleiben erhalten und werden nur ein bisschen zu den anderen geschoben.)

In der Layergestaltung beider Quellebenen (Hochgebirge und Mittelgebirge) wählen wir nun als Füllung *Geometriegenerator*, als Geometrietyp *Punkte* und als Einheiten – im vorliegenden Beispiel – *Millimeter*. Neben dem Ausdrucksfeld drücken wir den  $\epsilon$ -Knopf, um den Ausdrucksdialog zu öffnen. Dort wählen wir zunächst den Funktionseditor, erstellen zwei neue Funktions-Dateien namens „Zufallspunkte hoch“ und „Zufallspunkte mittel“ (oder beliebigen anderen Namen) und kopieren den Code der zwei Funktionen für Hoch- und Mittelgebirge hinein (s. Anhang). Dies können wir für beide Ebenen in einem erledigen – der Inhalt des Funktionseditors ist in beiden Ebenen verfügbar.

---

<sup>3</sup> Der ‚historische Kern‘ dieser Funktion ist ein Code, den Rob Jones auf <https://impermanent.io/2017/05/05/generative-pseudo-random-polygon-fill-patterns-in-qgis/> veröffentlicht hat (zuletzt abgerufen am 02.02.24). Ihm sei an dieser Stelle dafür gedankt.

<sup>4</sup> S. dazu unten VI.4.



Nun wechseln wir zum Reiter *Ausdruck*, und nun wird es ebenenspezifisch: Hier kommt der Ausdruck hinein, der die Funktion aufruft, sobald die Ebene sichtbar geschaltet wird. Fürs Erste mögen folgende Ausdrücke genügen, die ein ansprechendes Punktebild im Maßstab 1:15 000 000 erzeugen:

Zufallspunkte\_hoch(0.7, 0.35, 0.35, 33, 42, 0.25, \$scale)

Zufallspunkte\_mittel(0.53, 0.3, 0.48, 33, 42, 0.25, \$scale)

Man beachte hierbei, dass sich „Zufallspunkte\_hoch“ und „\_mittel“ nicht auf die soeben vergebenen Dateinamen beziehen, sondern auf die im Code festgelegten Funktionsnamen (s. Anhang). Die Parameter in den Klammern haben im oberen der beiden Beispiele folgende Bedeutung (s. dazu auch die Kommentierung zu Beginn beider Funktionen im Anhang):

- „0.7“ und „0.35“ sind Breite und Höhe der einzelnen Gitter-Kästchen, gemessen in Karteneinheiten, hier also Grad. Nimmt man für die Breite das Doppelte der Höhe, ergibt dies in europäischen Breiten ein ungefähr quadratisches Kästchen.
- Nochmals „0.35“ bestimmt das Maß der Zufallsstreuung, mit welcher der einzelne Punkt im jeweiligen Kästchen platziert wird.
- „33“ und „42“ sind beliebig wählbare Werte als Zufallssaaten für die Punktestreuung und für die Zuweisung der SVG-Dateinamen. So lange diese „Saat“ gleich bleibt, bleibt auch das Zufallsbild gleich. Wenn einem also das Hügelbild nicht gefällt, kann man an diesen Stellen variieren, ohne dass sich die Dichte der Punkte grundsätzlich ändert.
- „0.25“ bestimmt die Breite des Puffers, der um die Geometrie der erwähnten anderen Ebenen gezogen wird, mithin die von Punkten freizuhaltende Zone.
- Als „\$scale“ wird der Kehrwert des aktuellen Maßstabs übergeben (beim Maßstab 1:15 000 000 also die „15 000 000“). Damit können die o.g. Ausdrücke später so verfeinert werden, dass sie Rücksicht auf den jeweiligen Maßstab nehmen (s. im Folgenden).

## V. Generieren der Punkte für verschiedene Maßstäbe

Das Erzeugen der Punkte geschieht nun schlicht dadurch, dass man eine der Quellebenen sichtbar schaltet und damit die Funktion von QGIS ausführen lässt. Das kann eine Weile dauern, und es kann gut sein, dass QGIS dabei abstürzt! Mit etwas Glück sind die Punkte dann aber schon generiert und abgespeichert.

Das Generieren erfolgt zwar aus den Quellebenen heraus mittels der dort hinterlegten Funktionen, doch abgespeichert werden die Punkte in der Zielebene „Gebirgspunkte“. Die Quellebenen müssen und sollten also nur zum Zwecke der Punkterzeugung sichtbar geschaltet und dann gleich wieder unsichtbar gemacht werden.

Beide Funktionen sind so programmiert, dass sie nur dann Punkte erzeugen, wenn für den aktuellen Maßstab noch keine vorhanden sind. Sie sind auch so voreingestellt, dass nicht für jeden beliebigen, krummen Maßstab Punkte erzeugt werden, sondern nur für Maßstäbe, deren Kehrwert durch 250 000 teilbar ist, was z.B. auf den hier gewählten Ausgangsmaßstab von 1:15 000 000 und viele seiner Halbierungen zutrifft<sup>5</sup>. (Das lässt sich natürlich bei Bedarf in den Funktionen ändern, s. Anhang).

Man erzeugt also die Punkte für verschiedene Maßstäbe, indem man in verschiedenen Maßstäben die Quellebene sichtbar macht oder indem man bei sichtbarer Quellebene durch die verschiedenen Maßstäbe scrollt. Dabei zeigt sich sofort das Problem, dass die als Parameter übergebenen Abstände absolut sind, so dass die Gitterkästchen und damit auch die Punkte in jedem Maßstab denselben geografischen Abstand haben. Will man die Punkte bei doppeltem Maßstab doppelt so dicht haben, damit der optische Abstand der Gleiche bleibt, darf darum bei doppeltem Maßstab nur der halbe Wert als Parameter übergeben werden. Das erreicht man, indem man statt eines absoluten Werts einen Bruch unter Verwendung von „\$scale“ im Zähler einsetzt. (Zu genau diesem Zweck wird der Funktion „\$scale“ als letzter Parameter übergeben.) Wenn einem also z.B. im Maßstab 1:15 000 000 ein horizontaler Abstand von 0,7 passend erschien – also so:

Zufallspunkte\_hoch(0.7, ...,

dann kann man diese 0,7 durch einen Bruch ersetzen, der im genannten Maßstab dasselbe ergibt wie 0,7 – nämlich  $15\,000\,000 / 21\,428\,571,4$  – und der unter Verwendung von „\$scale“ statt „15 000 000“ sowie ein bisschen gerundet so lautet:

Zufallspunkte\_hoch(\$scale/21400000, ....

Bei doppeltem Maßstab ist dann „\$scale“ nur noch halb so groß, womit der geografische Abstand halbiert wäre, so dass er optisch der Gleiche bleibt. Will man nun, dass bei doppeltem Maßstab der optische Abstand doch nicht ganz gleich bleibt, sondern doch wieder ein klein bisschen größer wird, so kann man den Parameter noch um einen geringen Summanden ergänzen, der genau umgekehrt wirkt, z.B.:

Zufallspunkte\_hoch(\$scale/21400000+1200000/\$scale, ....

Entsprechend kann man mit allen sonstigen Parametern verfahren, die Abstände bezeichnen. Am Ende kamen bei mir folgende Ausdrücke heraus:

---

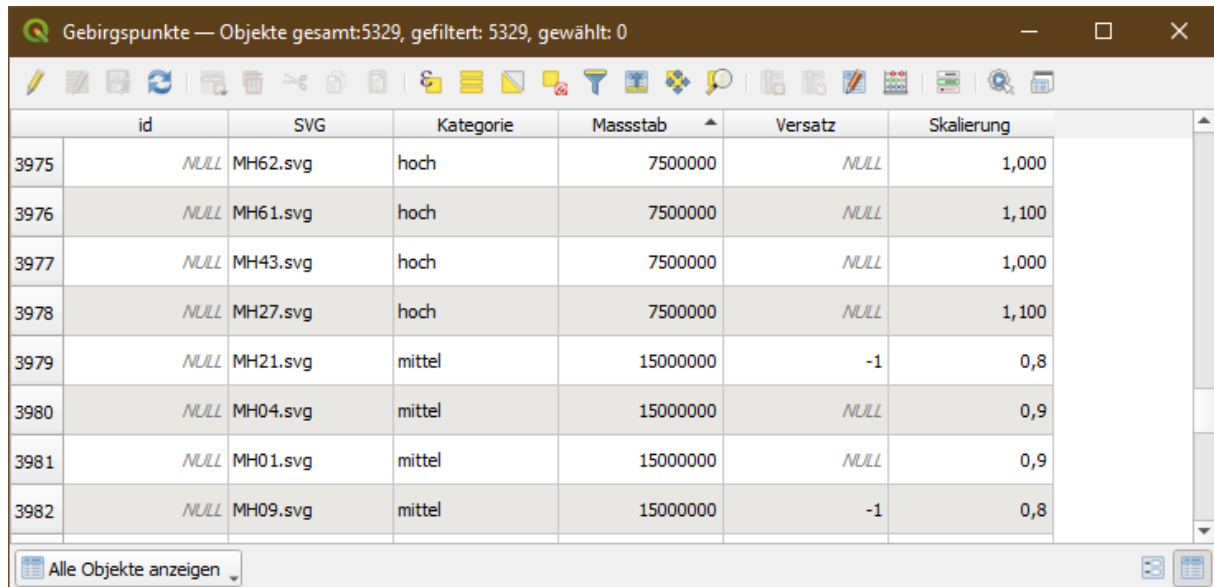
<sup>5</sup> Denn standardmäßig verdoppelt bzw. halbiert QGIS den Maßstab ja durch Scrollen, was allerdings unter *Einstellungen* → *Optionen* → *Kartenwerkzeuge* → *Zoomen* auch geändert werden kann.



Zufallspunkte\_hoch( $\$scale/25000000+1200000/\$scale$ ,  
 $\$scale/50000000+600000/\$scale$ ,  $\$scale/70000000+2000000/\$scale$ , 33, 42,  
 $\$scale/70000000+500000/\$scale$ ,  $\$scale$ )

Zufallspunkte\_mittel( $\$scale/33000000+1200000/\$scale$ ,  
 $\$scale/65000000+1000000/\$scale$ ,  $\$scale/60000000+3500000/\$scale$ , 33, 42,  
 $\$scale/70000000+500000/\$scale$ ,  $\$scale$ )

## VI. Darstellung der Punkte mit Hugelgrafiken



Gebirgspunkte — Objekte gesamt:5329, gefiltert: 5329, gewhlt: 0

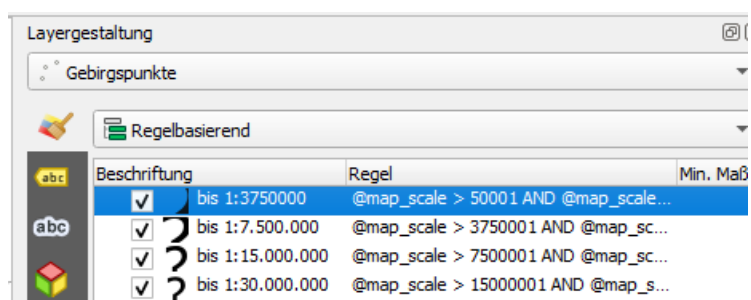
	id	SVG	Kategorie	Maßstab	Versatz	Skalierung
3975	NULL	MH62.svg	hoch	7500000	NULL	1,000
3976	NULL	MH61.svg	hoch	7500000	NULL	1,100
3977	NULL	MH43.svg	hoch	7500000	NULL	1,000
3978	NULL	MH27.svg	hoch	7500000	NULL	1,100
3979	NULL	MH21.svg	mittel	15000000	-1	0,8
3980	NULL	MH04.svg	mittel	15000000	NULL	0,9
3981	NULL	MH01.svg	mittel	15000000	NULL	0,9
3982	NULL	MH09.svg	mittel	15000000	-1	0,8

Alle Objekte anzeigen

Alle generierten Punkte sind nun in die Attributtabelle der Ebene ‚Gebirgspunkte‘ eingetragen, wo sie sich in ihren jeweiligen Attributen unterscheiden: In der Spalte „SVG“ ist hinterlegt, welche der 63 Maulwurfshugel-Dateien verwendet werden sollen. Aus der Spalte „Kategorie“ ergibt sich, ob es sich um einen Punkt fur das Hoch- oder fur das Mittelgebirge handelt. Unter „Maßstab“ steht, fur welchen Maßstab der Punkt eingetragen wurde – nur fur diesen soll er spater auch sichtbar sein. „Versatz“ weist fur manche Punkte einen moglichen Versatz nach Ost oder West aus: So liegen z.B. die oben mit „-1“ versehenen Punkte zu nahe am Westufer eines Flusses und sollten darum noch ein Stuck weiter nach Westen verschoben werden. Aus der letzten Spalte schließlich ergibt sich ein Skalierungsfaktor, der fur zwischen Flussen ‚eingeklemmte‘ Punkte (s. Anhang) „0,5“ betragt, fur zu verdrangende Punkte (s. Spalte ‚Versatz‘) „0,8“ und fur alle ubrigen Punkte einen zufalligen Wert von 0,9 bis 1,1 (in Zehntelschritten. Bei Bedarf kann dies im Funktionscode geandert werden, s. Anhang.)

### 1. Maßstabsabhangige Sichtbarkeit der einzelnen Punkte

Sorgen wir also zunachst dafur, dass jeder Punkt nur fur denjenigen Maßstab angezeigt wird, fur den er erstellt wurde, oder, genauer gesagt, fur einen Bereich von diesem Maßstab bis zum nachsten. Dafur bietet sich die *regelbasierende Symbolisierung* an:



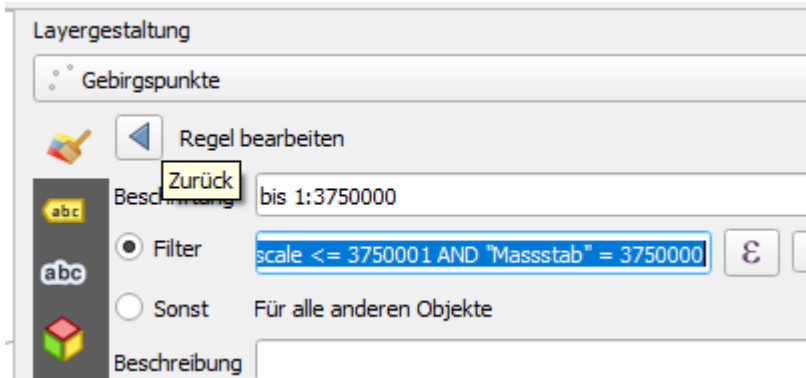
Layergestaltung

° Gebirgspunkte

Regelbasierend

ab	bis	Regel	Min. Maß
✓	bis 1:3750000	@map_scale > 50001 AND @map_scale...	
✓	bis 1:7.500.000	@map_scale > 3750001 AND @map_sc...	
✓	bis 1:15.000.000	@map_scale > 7500001 AND @map_sc...	
✓	bis 1:30.000.000	@map_scale > 15000001 AND @map_s...	

Im vorliegenden Beispiel wurden Punkte für die vier Maßstäbe 1:3750000, 1:7500000, 1:15000000 und 1:30000000 generiert. Dementsprechend definieren wir vier Anzeige-Regeln für die Bereiche von einem Maßstab bis zum nächsten, d.h. zunächst von einer willkürlich gegriffenen Untergrenze von 50001 bis zum Maßstabs-Kehrwert von 3750000, so dann von 3750000 bis 7500000 usw. Für jede dieser Regeln wird unter *Filter* ein Ausdruck eingetragen, der die Sichtbarkeit der Punkte regelt. Für die Regel „bis 1:3750000“ lautet er z.B.:



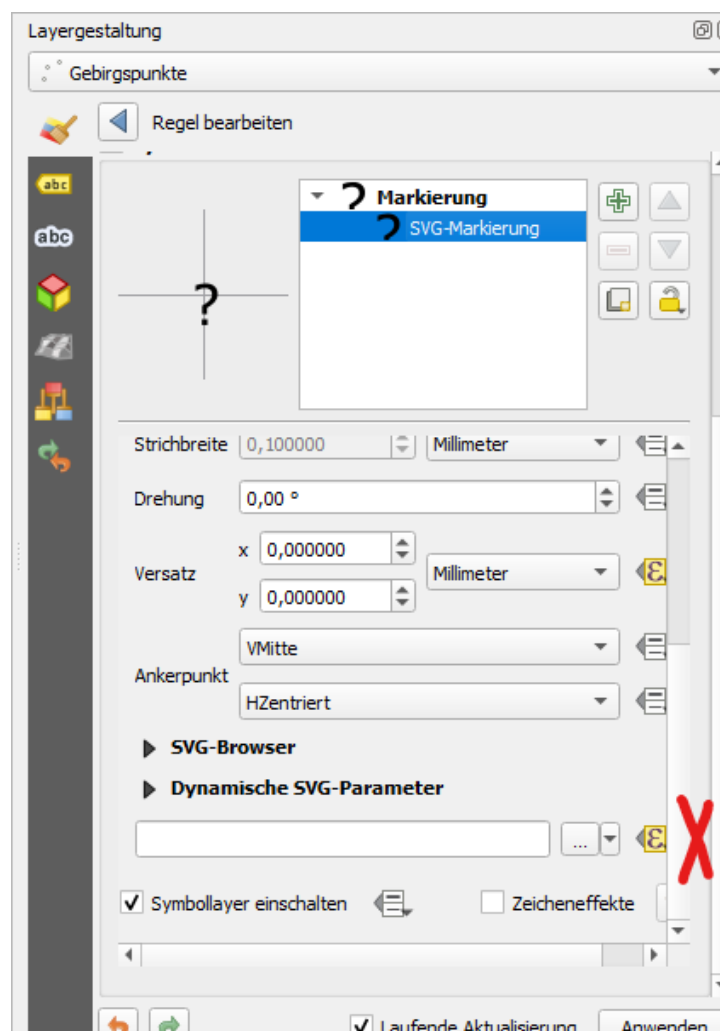
@map\_scale > 50001 AND @map\_scale <= 3750000 AND "Massstab" = 3750000

Das bedeutet: Ist der aktuelle Maßstabskehrwert größer als 50001 und zugleich kleiner od. gleich 3750000, und ist für den jeweiligen Punkt unter „Massstab“ „3750000“ eingetragen, so wird dieser Punkt angezeigt; wenn nicht, dann nicht.

Entsprechend wären auch die Regeln für die anderen Maßstabs-Bereiche zu formulieren.

## 2. Zuweisung der SVG-Dateien

Wenn wir doppelt auf die einzelne Regel klicken und dann noch einmal auf *SVG-Markierung*, so findet sich ganz unten ein Feld, wo der Pfad zu einer SVG-Datei eingetragen werden kann. Da wir jedoch nicht ein und dieselbe Hügelgrafik für sämtliche Punkte verwenden, sondern jedem Punkt die Grafik zuweisen wollen, die für ihn in der Attributtabelle hinterlegt ist, müssen wir stattdessen rechts von diesem Feld den ε-Knopf drücken, um einen Ausdruck wie den folgenden eintragen zu können:



'd:/Kartografie/Signaturen/Maulwurfshügel/' || "SVG"

Er gibt zunächst den Pfad zur SVG-Datei wieder, ergänzt um dasjenige, was in der Spalte „SVG“ steht, also den Dateinamen, – mithin insgesamt einen vollständigen Pfad wie zum Beispiel „D:\Kartografie\Signaturen\Maulwurfshügel\MH31.svg“.

### 3. Skalierung der Hugelgrafiken abhangig vom Mastab – und vom Breitengrad

Nach Doppelklick auf eine der Regeln und einfachem Klick auf *SVG- Markierung* finden wir unten auch die Felder, um die Groe der Hugelgrafiken festzulegen. Hier sollte zunachst eine passende Hugelgroe fur den nominalen Mastab der jeweiligen Regel eingestellt werden. (Wir sollten also darauf achten, dass die Karte sich aktuell in diesem Mastab befindet und nicht irgendwo zwischen diesem und dem nachsten.) Die angegebene Groe bezieht sich dabei auf den unsichtbaren Kreis, der im Hintergrund einer jeden Hugelgrafik liegt.

Wenn wir ein ansprechendes Kartenbild gefunden haben, sollten wir dafur sorgen, dass dieses auch dann erhalten bleibt, wenn die Karte in den Bereich zwischen diesem und dem nachsten Mastab hinein vergroert wird. Die Hugel sollten dabei namlich mit vergroert werden, um ihre relative Groe im Verhaltnis zum ubrigen Kartenbild zu behalten. Das erreichen wir, indem den  $\epsilon$ -Knopf drucken und mit *Bearbeiten* den Ausdruckseditor offnen, wo folgender Ausdruck eingetragen werden kann:

CASE

```
WHEN "Kategorie" = 'hoch' THEN 8 * "Skalierung" * 7500000/$scale * (-0.021 * $y + 2.1)
WHEN "Kategorie" = 'mittel' THEN 7.5 * "Skalierung" * 7500000/$scale * (-0.021 * $y + 2.1)
ELSE 1
```

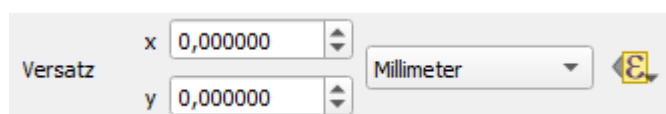
END

Seine ersten Zeilen bedeuten: Im Fall wenn die Kategorie „hoch“ ist, dann hat der Punkt eine Groe von 8 mm, die jedoch skaliert wird um den Faktor aus der Spalte „Skalierung“ (damit das Hugelbild abwechslungsreicher wird), anschließend um einen mastabsabhangigen Faktor, der im Nominal-Mastab genau 1 betragt, sich jedoch beim Vergroern des Mastabs mit vergroert, und schließlich um den hinteren, eingeklammerten Faktor, welcher den y-Wert auf der Karte berucksichtigt, also den Breitengrad. Denn da die Koordinatenbezugssysteme der vorliegenden Ebenen Grad als Karteneinheit haben, sind auch die Hugel-Abstande in Grad angegeben – einer Einheit also, die umso langer ist, je naher man dem Aquator kommt. Das Kartenbild wird darum etwas schoner, wenn die Hugel nach Suden hin im gleichen Mae groer werden wie die Grade. Bei dem hier verwendeten Koordinatenbezugssystem EPSG:3034 mit seiner konischen Lambert-Projektion divergieren die Meridiane in sudlicher Richtung linear, was es einfach macht, eine Formel fur eine ebenso lineare Vergroerung der Hugel zu finden (namlich „ $-0.021 * \$y + 2.1$ “. Das absolute Glied wurde hierbei so gewahlt, dass die Hugel etwa beim 50. Breitengrad ihre eigentliche Groe haben und von dort aus nordwarts kleiner und sudwarts groer werden.)

Fur die Mittelgebirgshugel verhalt es sich in der dritten Zeile entsprechend; sie erhalten eine grundsatzliche Groe von 7,5 mm. Der Anfangswert von 1 in der vorletzten Zeile wird nur relevant, falls unter „Kategorie“ nichts eingetragen ist (also eigentlich nie).

### 4. Abrucken der Hugel von Flussen, Kustenlinien etc.

Ebenfalls an der schon mehrfach erwahnten Stelle findet sich die Rubrik *Versatz*, wo anstatt eines festen Versatzes fur die Hugelgrafiken auch einer solcher eingestellt werden kann, der sich nach den Eintragen in der Attributspalte „Versatz“ richtet. Dafur drucken wir wieder den  $\epsilon$ -Knopf, gehen auf *Bearbeiten* und geben im Ausdruckseditor z.B. Folgendes ein:



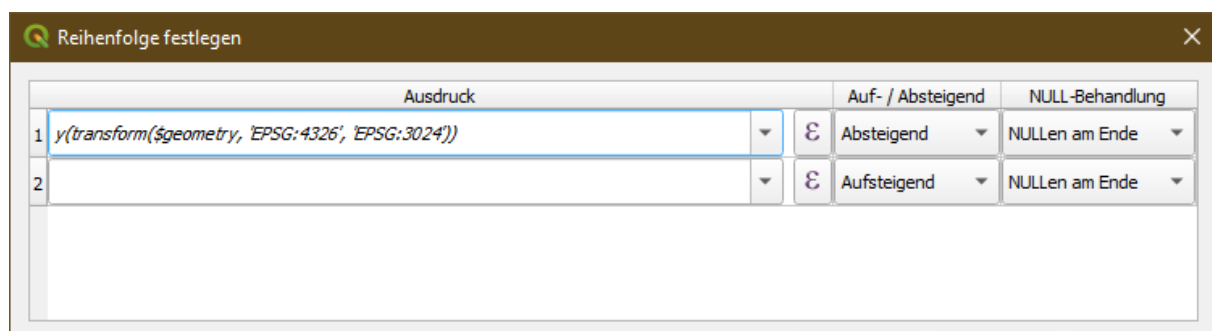
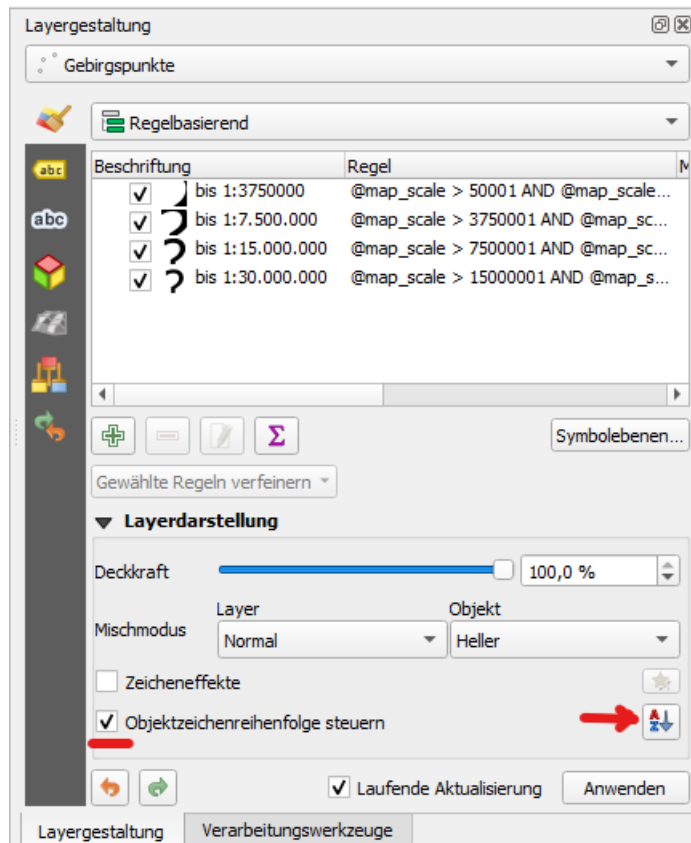
```
to_string("Versatz"*0.7) || ',' || 0
```

Dieser Ausdruck gibt einen Versatz im Format „x,y“ wieder. Als x-Wert wird dabei der Eintrag aus der Spalte „Versatz“ genommen und mit 0,7 multipliziert (einem Faktor, der frei wählbar ist). Das Ergebnis wird in einen ‚String‘ umgewandelt, um für QGIS verdaulich zu sein. Fort fährt die Verkettung mit einem Komma als Trennzeichen und einer Null als y-Wert.

## 5. Justieren der Zeichenreihenfolge fürs aktuelle Koordinatenreferenzsystem

Damit die Hügeldarstellung einen perspektivischen Eindruck erweckt, sind die beiden Funktionen schon so formuliert, dass die Hügel nach ihrem y-Wert sortiert und die südlicheren Hügel erst nach den nördlicheren gezeichnet werden, damit die vorderen Hügel die hinteren verdecken und nicht umgekehrt (s. Anhang). Damit sieht das Kartenbild schon halbwegs ordentlich aus. Als y-Wert dient hier aber der geografische Nord-/Süd-Wert, und dies führt dort zu unzureichenden Ergebnissen, wo Norden in der Karte nicht (oder nicht ganz) oben ist: nämlich in den östlichen und westlichen Randbereichen, wo die Meridiane schräg zum Nordpol hin verlaufen, und erst recht in Karten, die gar nicht genordet sind, sondern z.B. geostet (wie es in alten Zeiten nicht unüblich war).

Um ein wirklich ordentliches Kartenbild zu erhalten, müssen wir also erreichen, dass die Hügel in der Karte wirklich von oben nach unten gezeichnet werden. Das geht, indem wir in der *Layergestaltung* ganz unten das Häkchen bei *Objektzeichenreihenfolge steuern* setzen und rechts davon den Knopf drücken. Nun öffnet sich folgendes Fenster,



in welches wir folgenden Ausdruck eintragen können:

`y(transform($geometry, 'EPSG:4326', 'EPSG:3024'))`

Er bewirkt, dass die Punkte in der Reihenfolge ihres y-Werts gezeichnet werden, jedoch erst, nachdem die Koordinaten der Ebene (EPSG:4326) in die Koordinaten am Bildschirm (EPSG:3024) transformiert wurden.

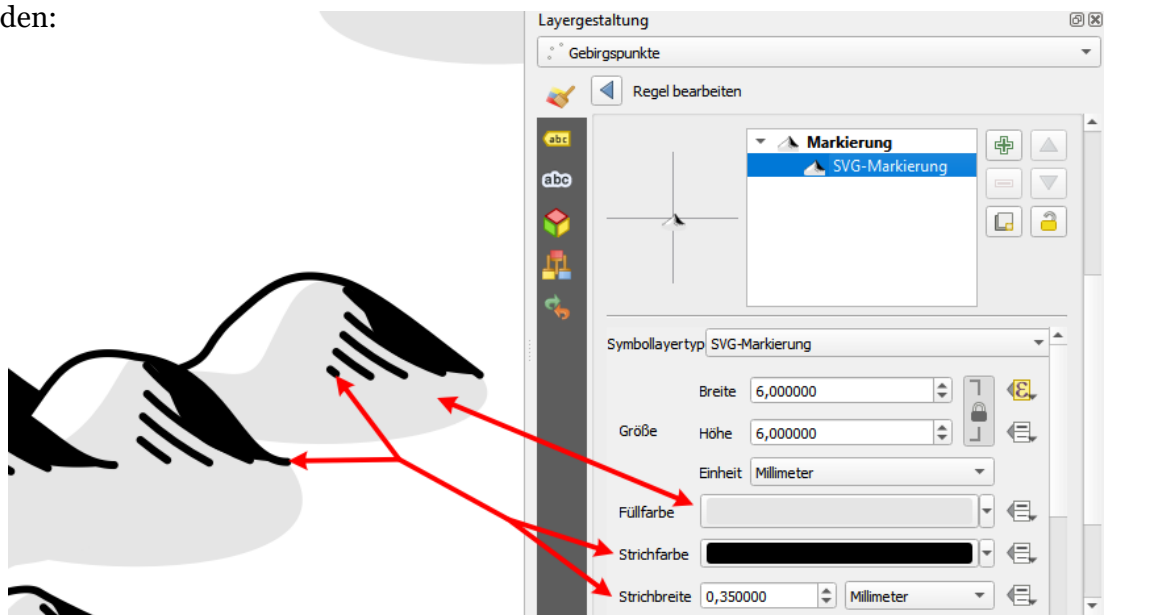


## 6. Einstellen von Farbe und Strichstärke der Hügel

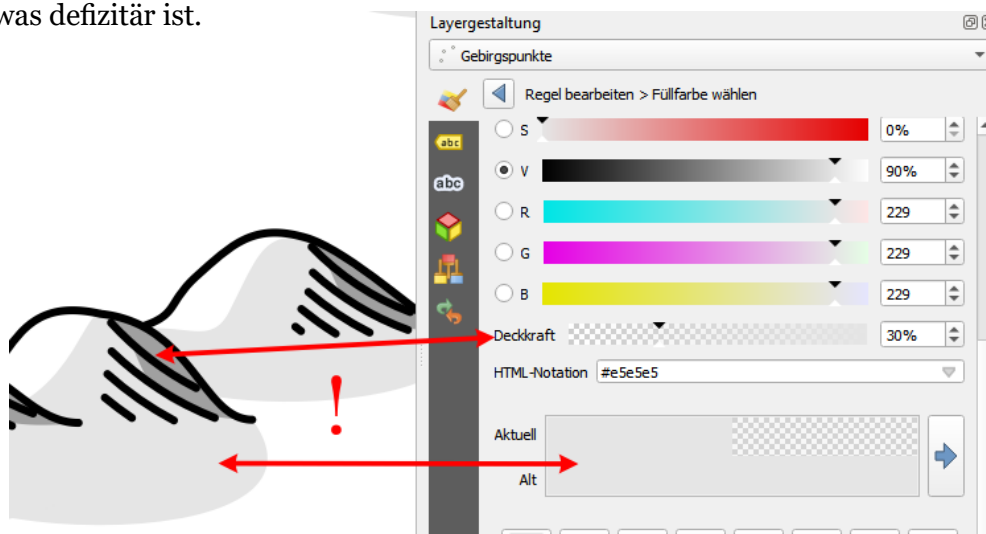
Zum Schluss geht es noch darum, Farben und Strichstärken der Hügelgrafiken einzustellen. Diese sind in den SVG-Dateien nämlich nicht fix hinterlegt, sondern als Variablen (Parameter) offen gelassen, deren Wert durch QGIS bestimmt werden kann (sog. ‚parametrisierte‘ SVG-Dateien). Das bedeutet, dass wir in QGIS das Aussehen der Hügelgrafiken an die Bedürfnisse unseres Kartenbildes anpassen können.

Um die nötigen Felder zu aktivieren, die in QGIS bisher gegraut sind, müssen wir allerdings doch noch so tun, als wollten wir ein und dieselbe Grafik für alle Hügel verwenden und an der auf S.10 angekreuzten Stelle den Pfad zu einer konkreten Hügeldatei angeben:

(Welche wir wählen, ist egal, da es sich nicht auswirkt, sondern nach wie vor datendefiniert übersteuert wird.) Nun können Farben und Strichstärken mit folgenden Feldern eingestellt werden:

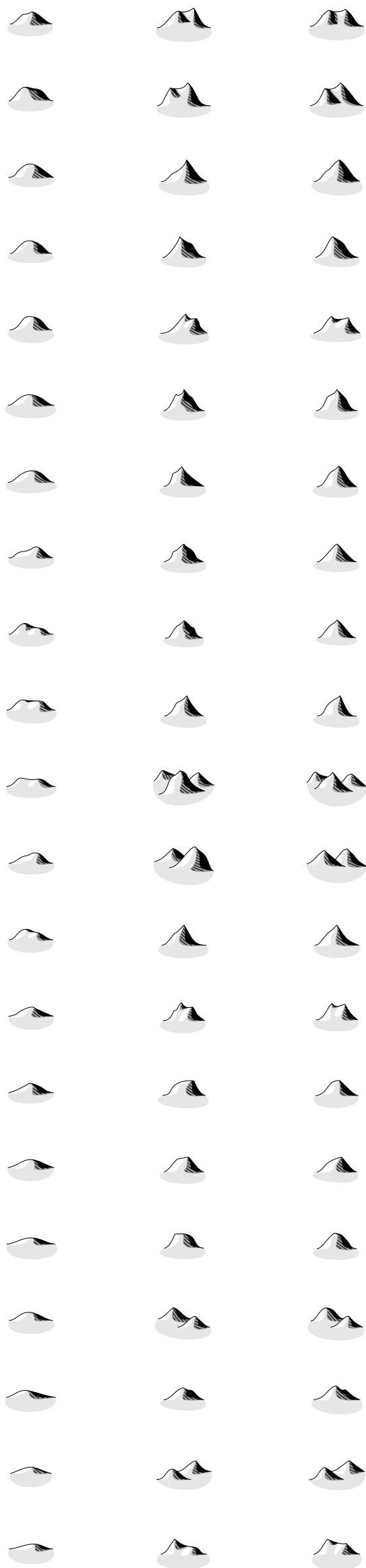


Klickt man auf das Farbfeld bei *Füllfarbe*, eröffnen sich genauere Einstellungsmöglichkeiten, u.a. hinsichtlich der Deckkraft. Hier ist zu beachten, dass QGIS die Deckkraft der Hügel Farbe einzustellen glaubt, während es sich in Wahrheit auf den schwarzen Schatten auswirkt! Diese Zweckentfremdung war notwendig, weil QGIS' Nutzung parametrisierter SVG-Dateien derzeit noch etwas defizitär ist.



## 63 ‚Maulwurfshügel‘ zur Verwendung auf Landkarten

### 63 ‘Molehills’ for use in maps



Hinweis: Damit diese Datei mit dem Skript ‚Maulwurfshügel\_zerlegen.py‘ aufgeteilt und die einzelnen Hügel als Dateien ‚MH01.svg‘ bis ‚MH63.svg‘ abgespeichert werden können, müssen die unsichtbaren Kreise im Hintergrund der Hügel unverändert bleiben!

Note: To split this file with the script ‘Maulwurfshügel\_zerlegen.py’ and save the individual hills as files ‘MH01.svg’ to ‘MH63.svg’, the invisible circles in the background of the hills must remain unchanged!

## Anhang: Funktionen und Skripte nebst Erläuterung

### Zufallspunkte\_hoch.py

Dies ist die Funktion zur Generierung der Punkte für die Hochgebirgs-Hügel, zu verwenden in der Quellebene „Hochgebirgspunkte“.

```
from qgis.core import *
import math
import random

# Definiere eine benutzerdefinierte Funktion für QGIS
@qgsfunction(args='auto', group='Custom')

def Zufallspunkte_hoch(xInterval, yInterval, rand, position_seed, filename_seed, buffer_distance, scale, feature,
parent):

    # Initialisiere die Zufallszahlengenerierung mit einem Saatwert für reproduzierbare Ergebnisse
    random.seed(position_seed)

    # Berechne die Begrenzungsbox des aktuellen Features
    box = feature.geometry().boundingBox()

    # Berechne, wie viele Punkte basierend auf den Intervallen in x- und y-Richtung passen
    countX = math.ceil(box.width() / xInterval)
    countY = math.ceil(box.height() / yInterval)

    # Berechne die neue Größe des Gitters, um die Begrenzungsbox abzudecken
    gridX = countX * xInterval
    gridY = countY * yInterval
    dX = gridX - box.width()
    dY = gridY - box.height()
    xMin = box.xMinimum() - (dX / 2)
```

**Kommentiert [RP1]:** Name der Funktion. Hierauf bezieht sich der Aufruf im Ausdruckseditor.

```

yMin = box.yMinimum() - (dY / 2)

# Runde den übergebenen Maßstab auf die nächste ganze Zahl
masstab_attribute = round(float(scale))

# Überprüfe, ob in der Ebene ‚Gebirgspunkte‘ bereits Punkte mit demselben Maßstab und derselben Gebirgs-Kategorie
vorhanden sind
layer = QgsProject.instance().mapLayersByName('Gebirgspunkte')[0]
existing_features = layer.getFeatures(QgsFeatureRequest().setFilterExpression(f'Masstab = {masstab_attribute} AND
Kategorie = \'hoch\'))

# Nur fortfahren, wenn keine entsprechenden Punkte vorhanden sind und der Maßstab glatt teilbar ist
if len(list(existing_features)) > 0 or masstab_attribute % 250000 != 0:
    return None

# Generiere Punkte innerhalb der Begrenzungsbox basierend auf den angegebenen Intervallen
points = []
for xOff in range(countX + 1):
    for yOff in range(countY + 1):
        ptX = xMin + xOff * xInterval + rand * random.uniform(0, xInterval)
        ptY = yMin + yOff * yInterval + rand * random.uniform(0, yInterval)
        pt = QgsPointXY(ptX, ptY)
        points.append(pt)

# Behalte nur Punkte, die innerhalb der Geometrie des Gebirgs-Polygons liegen
points = [pt for pt in points if feature.geometry().contains(QgsGeometry.fromPointXY(pt))]

# Überprüfe und entferne Punkte, die sich innerhalb der Puffer anderer relevanter Ebenen befinden
for other_layer_name in ['Flüsse 50 Ausschnitt', 'Seen 50', 'Küstenlinie 50 Ausschnitt']:
    other_layer = QgsProject.instance().mapLayersByName(other_layer_name)[0]
    other_features = other_layer.getFeatures()

    for other_feature in other_features:
        other_geometry = other_feature.geometry().simplify(0.05) # Vereinfache die Geometrie zur Optimierung

```

**Kommentiert [RP2]:** Das ist nötig, weil QGIS den Maßstab manchmal nicht glatt, sondern mit krummen Nachkommastellen übergibt.

**Kommentiert [RP3]:** Angabe der Zielebene. Bei Bedarf ändern.

**Kommentiert [RP4]:** Sorgt dafür, dass Punkte nur für solche Maßstäbe generiert werden, deren Kehrwert durch 250000 teilbar ist. Bei Bedarf ändern.

**Kommentiert [RP5]:** Hier müssen alle Ebenen eingetragen werden, von deren Geometrie die Hügel sich fernhalten sollen.



```

        buffer_geometry = other_geometry.buffer(buffer_distance, 3) # Erstelle einen Puffer um die Geometrie
        points = [pt for pt in points if not buffer_geometry.contains(QgsGeometry.fromPointXY(pt))]

```

```

# Sortiere die Punkte nach ihrer y-Position, um einen Tiefeneffekt zu erzielen

```

```

points.sort(key=lambda p: p.y(), reverse=True)

```

```

# Weise jedem Punkt ein zufälliges SVG-Symbol zu

```

```

svg_attribute = []

```

```

random.seed(filename_seed)

```

```

for pt in points:

```

```

    svg_number = str(random.randint(22, 63)).zfill(2)

```

```

    svg_attribute.append(f'MH{svg_number}.svg')

```

```

# Weise jedem Punkt einen zufälligen Skalierungsfaktor zu

```

```

skalierungsfaktoren = []

```

```

for _ in points:

```

```

    # Erzeuge einen zufälligen Skalierungsfaktor in Zehntelschritten von 0.9 bis 1.1

```

```

    faktor = round(random.uniform(0.9, 1.1), 1)

```

```

    skalierungsfaktoren.append(faktor)

```

```

# Füge die generierten Punkte als neue Features zur Ebene hinzu

```

```

layer.startEditing()

```

```

for svg_name, skalierungsfaktor, point in zip(svg_attribute, skalierungsfaktoren, points):

```

```

    new_feature = QgsFeature(layer.fields())

```

```

    new_feature.setAttribute('SVG', svg_name)

```

```

    new_feature.setAttribute('Massstab', massstab_attribute)

```

```

    new_feature.setAttribute('Kategorie', 'hoch')

```

```

    new_feature.setAttribute('Skalierung', skalierungsfaktor) # Setze den Skalierungsfaktor

```

```

    new_feature.setGeometry(QgsGeometry.fromPointXY(point))

```

```

    layer.addFeature(new_feature)

```

```

# Identifiziere Punkte für die Versatzprüfung basierend auf einem erweiterten Puffer

```

```

vorsortierte_punkte = []

```

```

for other_layer_name in ['Flüsse 50 Ausschnitt', 'Seen 50', 'Küstenlinie 50 Ausschnitt']:

```

**Kommentiert [RP6]:** Sorgt dafür, dass die südlicheren Punkte vor den nördlicheren liegen. Ist ungenügend, wenn Nordrichtung nicht ganz senkrecht ist (z.B. in Randbereichen der Karte oder wenn Karte nicht genor-det ist); bedarf dann zusätzlicher Anpassung in QGIS.

**Kommentiert [RP7]:** Bezieht sich auf die Dateien „MH22.svg“ bis „MH63.svg“ – das sind die Symbole für die Hochgebirgs-Maulwurfshügel.

**Kommentiert [RP8]:** Bereich von 0,9–1,1, aus welchem den Punkten ein zufälliger Skalierungsfaktor (in Zehn-telschritten) zugewiesen wird. Bei Bedarf anzupassen.

**Kommentiert [RP9]:** Hier nochmals Ebenen eintragen, von deren Geometrie sich die Hügel fernhalten sollen. (Ich habe darauf verzichtet, dies ‚vor die Klammer zu ziehen‘, damit bei Bedarf jeweils andere Ebenen einge-tragen werden können. Z.B. soll in der anderen Funk-tion weiter unten, nämlich der für die Mittelgebirge, der Haupt-Puffer auch um die Hochgebirgs-Ebene gezogen werden; bei den Versatz-Puffern scheint mir dies jedoch entbehrlich.)

```
other_layer = QgsProject.instance().mapLayersByName(other_layer_name)[0]
other_features = other_layer.getFeatures()
```

```
for other_feature in other_features:
    erweiterter_puffer = other_feature.geometry().buffer(1.5 * buffer_distance, 3)
    for pt in points:
        if erweiterter_puffer.contains(QgsGeometry.fromPointXY(pt)):
            vorsortierte_punkte.append(pt)
```

```
# Identifiziere ,eingeklemmte' Punkte
```

```
punkte_ohne_versatz = set()
for pt in vorsortierte_punkte:
    verschobener_pt_rechts = QgsPointXY(pt.x() + buffer_distance / 2, pt.y())
    verschobener_pt_links = QgsPointXY(pt.x() - buffer_distance / 2, pt.y())
    im_puffer_rechts = False
    im_puffer_links = False
```

```
for other_layer_name in ['Flüsse 50 Ausschnitt', 'Seen 50', 'Küstenlinie 50 Ausschnitt']:
    other_layer = QgsProject.instance().mapLayersByName(other_layer_name)[0]
    other_features = other_layer.getFeatures()
    for other_feature in other_features:
        original_puffer = other_feature.geometry().buffer(buffer_distance, 3)
        if original_puffer.contains(QgsGeometry.fromPointXY(verschobener_pt_rechts)):
            im_puffer_rechts = True
        if original_puffer.contains(QgsGeometry.fromPointXY(verschobener_pt_links)):
            im_puffer_links = True
```

```
if im_puffer_rechts and im_puffer_links:
    punkte_ohne_versatz.add(pt)
```

```
# Setze Skalierung auf 0.5 für eingeklemmte Punkte, ohne Versatz zu ändern
```

```
for pt in punkte_ohne_versatz:
    for feature in layer.getFeatures():
        if feature.geometry().asPoint() == pt:
```

**Kommentiert [RP10]:** Identifiziert Hügel, die beidseits von Flüssen etc. bedrängt werden, also zwischen diesen eingeklemmt sind.

**Kommentiert [RP11]:** Hier nochmals die anderen Ebenen eintragen.

**Kommentiert [RP12]:** Für eingeklemmte Hügel wird kein Versatz nach Ost oder West eingetragen; stattdessen werden sie deutlich verkleinert.

```

        feature.setAttribute('Skalierung', 0.5)
        layer.updateFeature(feature)

# Prüfe vorsortierte Punkte auf Versatz nach rechts und setze entsprechende Attribute
for pt in vorsortierte_punkte:
    if pt in punkte_ohne_versatz:
        continue # Überspringe Punkte ohne Versatz
    verschobener_pt = QgsPointXY(pt.x() + buffer_distance / 2, pt.y())

    for other_layer_name in ['Flüsse 50 Ausschnitt', 'Seen 50', 'Küstenlinie 50 Ausschnitt']:
        other_layer = QgsProject.instance().mapLayersByName(other_layer_name)[0]
        other_features = other_layer.getFeatures()

        for other_feature in other_features:
            original_puffer = other_feature.geometry().buffer(buffer_distance, 3)
            if original_puffer.contains(QgsGeometry.fromPointXY(verschobener_pt)):
                for feature in layer.getFeatures():
                    if feature.geometry().asPoint() == pt and feature not in punkte_ohne_versatz:
                        feature.setAttribute('Versatz', -1)
                        feature.setAttribute('Skalierung', 0.8)
                        layer.updateFeature(feature)
                        break

# Prüfe vorsortierte Punkte auf Versatz nach links und setze entsprechende Attribute
for pt in vorsortierte_punkte:
    if pt in punkte_ohne_versatz:
        continue # Überspringe Punkte ohne Versatz
    verschobener_pt_links = QgsPointXY(pt.x() - buffer_distance / 2, pt.y())

    for other_layer_name in ['Flüsse 50 Ausschnitt', 'Seen 50', 'Küstenlinie 50 Ausschnitt']:
        other_layer = QgsProject.instance().mapLayersByName(other_layer_name)[0]
        other_features = other_layer.getFeatures()

        for other_feature in other_features:

```

**Kommentiert [RP13]:** Hier nochmals die anderen Ebenen eintragen.

**Kommentiert [RP14]:** Hügel, die zu nah westlich neben einem Fluss etc. liegen, werden etwas westwärts abgerückt und geringfügig verkleinert.

**Kommentiert [RP15]:** Hier nochmals die anderen Ebenen eintragen.

```

original_puffer = other_feature.geometry().buffer(buffer_distance, 3)
if original_puffer.contains(QgsGeometry.fromPointXY(verschobener_pt_links)):
    for feature in layer.getFeatures():
        if feature.geometry().asPoint() == pt and feature not in punkte_ohne_versatz:
            feature.setAttribute('Versatz', 1)
            feature.setAttribute('Skalierung', 0.8)
            layer.updateFeature(feature)
            break

# Schreibe Änderungen in Ebene
layer.commitChanges()

return QgsGeometry.fromMultiPointXY(points)

```

**Kommentiert [RP16]:** Hügel, die zu nah östlich neben einem Fluss etc. liegen, werden etwas ostwärts abgerückt und geringfügig verkleinert.

### Zufallspunkte\_mittel.py

Dies ist die Funktion zur Erzeugung der Punkte für die Mittelgebirgs-Hügel, zu verwenden in der Quellebene „Hochgebirgspunkte“.

Kommentiert werden hier nur noch die spezifischen Unterschied zur vorigen Funktion.

```

from qgis.core import *
import math
import random

# Definiere eine benutzerdefinierte Funktion für QGIS
@qgsfunction(args='auto', group='Custom')

def Zufallspunkte_mittel(xInterval, yInterval, rand, position_seed, filename_seed, buffer_distance, scale, feature,
parent):

    # Initialisiere die Zufallszahlengenerierung mit einem Saatwert für reproduzierbare Ergebnisse
    random.seed(position_seed)

    # Berechne die Begrenzungsbox des aktuellen Features
    box = feature.geometry().boundingBox()

```

**Kommentiert [RP17]:** Name der Funktion, maßgeblich für ihren Aufruf.



```

# Berechne, wie viele Punkte basierend auf den Intervallen in x- und y-Richtung passen
countX = math.ceil(box.width() / xInterval)
countY = math.ceil(box.height() / yInterval)

# Berechne die neue Größe des Gitters, um die Begrenzungsbox abzudecken
gridX = countX * xInterval
gridY = countY * yInterval
dX = gridX - box.width()
dY = gridY - box.height()
xMin = box.xMinimum() - (dX / 2)
yMin = box.yMinimum() - (dY / 2)

# Runde den übergebenen Maßstab auf die nächste ganze Zahl
massstab_attribute = round(float(scale))

# Überprüfe, ob in der Ebene ‚Gebirgspunkte‘ bereits Punkte mit demselben Maßstab und derselben Gebirgs-Kategorie
vorhanden sind
layer = QgsProject.instance().mapLayersByName('Gebirgspunkte')[0]
existing_features = layer.getFeatures(QgsFeatureRequest().setFilterExpression(f'Massstab = {massstab_attribute} AND
Kategorie = \'mittel\'))

# Nur fortfahren, wenn keine entsprechenden Punkte vorhanden sind und der Maßstab glatt teilbar ist
if len(list(existing_features)) > 0 or massstab_attribute % 250000 != 0:
    return None

# Generiere Punkte innerhalb der Begrenzungsbox basierend auf den angegebenen Intervallen
points = []
for xOff in range(countX + 1):
    for yOff in range(countY + 1):
        ptX = xMin + xOff * xInterval + rand * random.uniform(0, xInterval)
        ptY = yMin + yOff * yInterval + rand * random.uniform(0, yInterval)
        pt = QgsPointXY(ptX, ptY)
        points.append(pt)

```

```

# Behalte nur Punkte, die innerhalb der Geometrie des Gebirgs-Polygons liegen
points = [pt for pt in points if feature.geometry().contains(QgsGeometry.fromPointXY(pt))]

# Überprüfe und entferne Punkte, die sich innerhalb der Puffer anderer relevanter Layer befinden
for other_layer_name in ['Flüsse 50 Ausschnitt', 'Seen 50', 'Küstenlinie 50 Ausschnitt', 'Hochgebirge']:
    other_layer = QgsProject.instance().mapLayersByName(other_layer_name)[0]
    other_features = other_layer.getFeatures()

    for other_feature in other_features:
        other_geometry = other_feature.geometry().simplify(0.05) # Vereinfache die Geometrie zur Optimierung
        buffer_geometry = other_geometry.buffer(buffer_distance, 3) # Erstelle einen Puffer um die Geometrie
        points = [pt for pt in points if not buffer_geometry.contains(QgsGeometry.fromPointXY(pt))]

# Sortiere die Punkte nach ihrer y-Position, um einen Tiefeneffekt zu erzielen
points.sort(key=lambda p: p.y(), reverse=True)

# Weise jedem Punkt ein zufälliges SVG-Symbol zu
svg_attribute = []
random.seed(filename_seed)
for pt in points:
    svg_number = str(random.randint(1, 21)).zfill(2)
    svg_attribute.append(f'MH{svg_number}.svg')

# Weise jedem Punkt einen zufälligen Skalierungsfaktor zu
skalierungsfaktoren = []
for _ in points:
    # Erzeuge einen zufälligen Skalierungsfaktor in Zehntelschritten von 0.9 bis 1.1
    faktor = round(random.uniform(0.9, 1.1), 1)
    skalierungsfaktoren.append(faktor)

# Füge die generierten Punkte als neue Features zur Ebene hinzu
layer.startEditing()
for svg_name, skalierungsfaktor, point in zip(svg_attribute, skalierungsfaktoren, points):

```

**Kommentiert [RP18]:** Hier ist auch die Ebene ‚Hochgebirge‘ aufgeführt, damit die Mittelgebirgshügel etwas Abstand von den Hochgebirgshügeln halten.

**Kommentiert [RP19]:** Bezieht sich auf die Dateien „MH22.svg“ bis „MH21.svg“ – das sind die Symbole für die Mittelgebirgs-Maulwurfshügel.

```

new_feature = QgsFeature(layer.fields())
new_feature.setAttribute('SVG', svg_name)
new_feature.setAttribute('Massstab', massstab_attribute)
new_feature.setAttribute('Kategorie', 'mittel')
new_feature.setAttribute('Skalierung', skalierungsfaktor) # Setze den Skalierungsfaktor
new_feature.setGeometry(QgsGeometry.fromPointXY(point))
layer.addFeature(new_feature)

# Identifiziere Punkte für die Versatzprüfung basierend auf einem erweiterten Puffer
vorsortierte_punkte = []
for other_layer_name in ['Flüsse 50 Ausschnitt', 'Seen 50', 'Küstenlinie 50 Ausschnitt']:
    other_layer = QgsProject.instance().mapLayersByName(other_layer_name)[0]
    other_features = other_layer.getFeatures()

    for other_feature in other_features:
        erweiterter_puffer = other_feature.geometry().buffer(1.5 * buffer_distance, 3)
        for pt in points:
            if erweiterter_puffer.contains(QgsGeometry.fromPointXY(pt)):
                vorsortierte_punkte.append(pt)

# Identifiziere ,eingeklemmte' Punkte
punkte_ohne_versatz = set()
for pt in vorsortierte_punkte:
    verschobener_pt_rechts = QgsPointXY(pt.x() + buffer_distance / 2, pt.y())
    verschobener_pt_links = QgsPointXY(pt.x() - buffer_distance / 2, pt.y())
    im_puffer_rechts = False
    im_puffer_links = False

    for other_layer_name in ['Flüsse 50 Ausschnitt', 'Seen 50', 'Küstenlinie 50 Ausschnitt']:
        other_layer = QgsProject.instance().mapLayersByName(other_layer_name)[0]
        other_features = other_layer.getFeatures()
        for other_feature in other_features:
            original_puffer = other_feature.geometry().buffer(buffer_distance, 3)
            if original_puffer.contains(QgsGeometry.fromPointXY(verschobener_pt_rechts)):

```

**Kommentiert [RP20]:** Dagegen scheint es bei der Prüfung auf seitlichen Versatz entbehrlich, auch die Hochgebirgs-Ebene zu berücksichtigen – deshalb fehlt diese hier und im Folgenden. (Das macht die Funktion komplizierter, aber die Berechnung ein bisschen einfacher.)

```

        im_puffer_rechts = True
        if original_puffer.contains(QgsGeometry.fromPointXY(verschobener_pt_links)):
            im_puffer_links = True

    if im_puffer_rechts and im_puffer_links:
        punkte_ohne_versatz.add(pt)

# Setze Skalierung auf 0.5 für Punkte ohne Versatz, ohne Versatz zu ändern
for pt in punkte_ohne_versatz:
    for feature in layer.getFeatures():
        if feature.geometry().asPoint() == pt:
            feature.setAttribute('Skalierung', 0.5)
            layer.updateFeature(feature)

# Prüfe vorsortierte Punkte auf Versatz nach rechts und setze entsprechende Attribute
for pt in vorsortierte_punkte:
    if pt in punkte_ohne_versatz:
        continue # Überspringe Punkte ohne Versatz
    verschobener_pt = QgsPointXY(pt.x() + buffer_distance / 2, pt.y())

    for other_layer_name in ['Flüsse 50 Ausschnitt', 'Seen 50', 'Küstenlinie 50 Ausschnitt']:
        other_layer = QgsProject.instance().mapLayersByName(other_layer_name)[0]
        other_features = other_layer.getFeatures()

        for other_feature in other_features:
            original_puffer = other_feature.geometry().buffer(buffer_distance, 3)
            if original_puffer.contains(QgsGeometry.fromPointXY(verschobener_pt)):
                for feature in layer.getFeatures():
                    if feature.geometry().asPoint() == pt and feature not in punkte_ohne_versatz:
                        feature.setAttribute('Versatz', -1)
                        feature.setAttribute('Skalierung', 0.8)
                        layer.updateFeature(feature)
                        break

```



```

# Prüfe vorsortierte Punkte auf Versatz nach links und setze entsprechende Attribute
for pt in vorsortierte_punkte:
    if pt in punkte_ohne_versatz:
        continue # Überspringe Punkte ohne Versatz
    verschobener_pt_links = QgsPointXY(pt.x() - buffer_distance / 2, pt.y())

    for other_layer_name in ['Flüsse 50 Ausschnitt', 'Seen 50', 'Küstenlinie 50 Ausschnitt']:
        other_layer = QgsProject.instance().mapLayersByName(other_layer_name)[0]
        other_features = other_layer.getFeatures()

        for other_feature in other_features:
            original_puffer = other_feature.geometry().buffer(buffer_distance, 3)
            if original_puffer.contains(QgsGeometry.fromPointXY(verschobener_pt_links)):
                for feature in layer.getFeatures():
                    if feature.geometry().asPoint() == pt and feature not in punkte_ohne_versatz:
                        feature.setAttribute('Versatz', 1)
                        feature.setAttribute('Skalierung', 0.8)
                        layer.updateFeature(feature)
                        break

# Schreibe Änderungen in Ebene
layer.commitChanges()

return QgsGeometry.fromMultiPointXY(points)

```

### Maulwurfshügel\_zerlegen.py

Wer die mitgelieferten Maulwurfshügel modifizieren möchte, tut dies am besten in der Datei „Maulwurfshügel.svg“, wo sie gesammelt beieinander sind. Anschließend können sie mithilfe dieses Python-Skripts hier in die 63 Einzeldateien „MH01.svg“ bis „MH63.svg“ aufgeteilt werden. Die einzelnen Hügel werden dabei anhand des unsichtbaren Kreises im Hintergrund eines jeden Hügel erkannt. Diese Kreise müssen darum bei der Bearbeitung der Hügel unbedingt unangetastet bleiben.

Ausgeführt wird dieses Skript einfach durch Doppelklick unter Windows, wobei es sich im selben Verzeichnis befinden muss wie die Datei „Maulwurfshügel.svg“.

```

# Dieses Skript zerlegt die Datei „Maulwurfshügel.svg“ in die einzelnen Dateien „MH01.svg“ bis „MH63.svg“.
Voraussetzung ist allerdings, dass sich im Hintergrund der 63 Hügel unverändert die 63 unsichtbaren Kreise mit den
Pfadnummern 1-63 befinden!
# This script splits the file "Maulwurfshügel.svg" into the individual files "MH01.svg" to "MH63.svg". However, the
prerequisite is that the 63 invisible circles with the path numbers 1-63 remain unchanged in the background of the 63
hills!

from xml.etree import ElementTree as ET
import re
import os
import shutil

def find_existing_svgs(base_dir):
    """
    Findet existierende MH##.svg Dateien im angegebenen Verzeichnis.
    """
    existing_svgs = [f for f in os.listdir(base_dir) if os.path.isfile(os.path.join(base_dir, f)) and
f.startswith("MH") and f.endswith(".svg") and f[2:4].isdigit()]
    return existing_svgs

def create_new_alt_directory(base_dir):
    """
    Erstellt ein neues alt##-Verzeichnis im angegebenen Basisverzeichnis.
    """
    existing_alts = [d for d in os.listdir(base_dir) if os.path.isdir(os.path.join(base_dir, d)) and
d.startswith("alt")]
    highest_num = 0
    for dir_name in existing_alts:
        try:
            num = int(dir_name[3:])
            highest_num = max(highest_num, num)
        except ValueError:
            continue
    new_dir_name = f"alt{highest_num + 1:02}"

```

```

    new_dir_path = os.path.join(base_dir, new_dir_name)
    return new_dir_name, new_dir_path

def move_svgs_to_alt_dir(svgs, alt_dir):
    """
    Verschiebt die gegebenen SVG-Dateien in das angegebene Verzeichnis.
    """
    for svg in svgs:
        shutil.move(svg, alt_dir)

# Überprüfe, ob zu sichernde SVG-Dateien existieren
base_dir = "." # Aktuelles Verzeichnis als Basisverzeichnis
existing_svgs = find_existing_svgs(base_dir)

if existing_svgs:
    # Erstelle ein neues alt##-Verzeichnis, wenn notwendig
    new_dir_name, new_dir_path = create_new_alt_directory(base_dir)
    os.makedirs(new_dir_path, exist_ok=True)
    # Verschiebe existierende MH##.svg Dateien
    move_svgs_to_alt_dir(existing_svgs, new_dir_path)
    print(f"SVG-Dateien nach {new_dir_name} verschoben.")
else:
    print("Keine SVG-Dateien zum Verschieben gefunden.")

# Die SVG-Namespace-Definition
SVG_NAMESPACE = "{http://www.w3.org/2000/svg}"
ET.register_namespace("", "http://www.w3.org/2000/svg")

# Der Dateiname der ursprünglichen SVG-Datei
original_file_name = "Maulwurfshügel.svg"

def extract_paths(svg_content):
    """
    Extrahiert Pfade aus der SVG-Datei und gruppiert sie nach ihren IDs.

```

```

"""
paths = {}
tree = ET.ElementTree(ET.fromstring(svg_content))
for path in tree.findall(f'://{SVG_NAMESPACE}path'):
    path_id = path.get('id')
    if path_id:
        paths[path_id] = path
return paths, tree

def calculate_bounding_box(path):
    """
    Berechnet die Bounding Box eines Pfades anhand seiner 'd'-Attribute.
    """
    d_attr = path.get('d')
    coords = list(map(float, re.findall(r"-?\d+\.\d*", d_attr)))
    x_coords = coords[::2]
    y_coords = coords[1::2]
    return min(x_coords), max(x_coords), min(y_coords), max(y_coords)

def path_in_bounding_box(path, bbox):
    """
    Vereinfachte Überprüfung, ob ein Knotenpunkt eines Pfades innerhalb einer gegebenen Bounding Box liegt.
    """
    d_attr = path.get('d')
    # Extrahiere den ersten Koordinatenpunkt
    coords = list(map(float, re.findall(r"-?\d+\.\d*", d_attr)[2:]))
    x, y = coords[0], coords[1]
    return bbox[0] <= x <= bbox[1] and bbox[2] <= y <= bbox[3]

def create_svg_with_custom_viewbox(paths, bbox, file_name):
    """
    Erstellt eine neue SVG-Datei mit den gegebenen Pfaden und einer angepassten viewBox.
    """
    viewBox_value = f"{bbox[0]} {bbox[2]} {bbox[1]-bbox[0]} {bbox[3]-bbox[2]}"

```

```

svg_root = ET.Element('svg', attrib={
    'xmlns:svg': "http://www.w3.org/2000/svg",
    'version': "1.1",
    'width': "12mm",
    'height': "12mm",
    'viewBox': viewBox_value
})
for path in paths:
    svg_root.append(path)
tree = ET.ElementTree(svg_root)
tree.write(file_name, encoding="utf-8", xml_declaration=True)

with open(original_file_name, 'r') as file:
    svg_content = file.read()

paths, original_tree = extract_paths(svg_content)

for path_id in range(1, 64):
    main_path_id = f'path{path_id}'
    main_path = paths.get(main_path_id)
    if main_path is None:
        continue
    bbox = calculate_bounding_box(main_path)
    # Überprüfung für die Zugehörigkeit basiert nun darauf, ob ein Punkt innerhalb der Bounding Box liegt
    included_paths = [main_path] + [path for pid, path in paths.items() if pid != main_path_id and
path_in_bounding_box(path, bbox)]

    new_file_name = f"MH{path_id:02}.svg"
    create_svg_with_custom_viewbox(included_paths, bbox, new_file_name)

print("SVG-Dateien wurden erstellt.")

```

## Maulwurfshügel\_parametrisieren.py

Mit diesem Skript lassen sich die SVG-Dateien der 63 Maulwurfshügel ‚parametrisieren‘. Das bedeutet, dass allerlei Werte für Farben und Strichstärken durch Variablen („Parameter“) ersetzt werden, deren Inhalt dann durch QGIS bestimmt werden kann. Zuvor wird eine Sicherungskopie der vorhandenen Hügeldateien in einem Unterverzeichnis namens „alto1“ oder ähnlich angelegt.

Auch dieses Skript wird einfach durch Doppelklick unter Windows ausgeführt, wobei es sich im selben Verzeichnis befinden muss wie die 63 Hügeldateien. Beachten Sie, dass es nicht dazu taugt, parametrisierte Maulwurfshügeldateien ‚umzuparametrisieren‘ – die Suchen-und-Ersetzen-Routinen dieses Skripts passen nur auf SVG-Dateien, die mit dem obigen Skript ‚Maulwurfshügel\_zerlegen.py‘ frisch erzeugt wurden.

```
import glob
import re
import os
import shutil

# Funktion, die die spezifischen Ersetzungen in einem gegebenen Text durchführt
def replace_content(text):
    replacements = {
        r'style="fill-rule: evenodd; fill: #e6e6e6; ': r'style="fill-rule: evenodd; fill: param(fill) #ffffff; fill-
opacity:1"', #Grundfarbe
        r'style="fill-rule: evenodd; fill: #999999; ': r'style="fill-rule: evenodd; fill: #000000; fill-opacity: 0"',
#Schatten 2
        r'style="fill-rule: evenodd; fill: #ffffff; ': r'style="fill-rule: evenodd; fill: #ffffff; fill-opacity: 1"',
#Licht
        r'style="stroke-width: 0.2160; stroke: #000000; stroke-dasharray: none; stroke-linecap: round; stroke-linejoin:
miter; fill: none; ': r'style="stroke-width: param(outline-width) 0.25; stroke: param(outline) #000000; stroke-
dasharray: none; stroke-linecap: round; stroke-linejoin: miter; fill: none"', #Schraffen
        r'style="fill-rule: evenodd; fill: #000000; ': r'style="fill-rule: evenodd; fill: #000000; fill-
opacity: param(fill-opacity) 1"', # Schatten 1
        r'style="stroke-width: 0.3399; stroke: #000000; stroke-dasharray: none; stroke-linecap: round; stroke-linejoin:
round; fill: none; ': r'style="stroke-width: param(outline-width) 0.35; stroke: param(outline) #000000; stroke-
dasharray: none; stroke-linecap: round; stroke-linejoin: round; fill: none"', #Außenlinie
        r'<path d=" ' : r'<path d="',
        r'; ': r';',
        r'" />': r'"/>',
```

**Kommentiert [RP21]:** Diese Variable, d.h. der Parameter „param(fill)“ gestattet es, die Füllfarbe für den Hügelhang nicht fest in der SVG-Datei zu hinterlegen, sondern in QGIS einzustellen. Die darauffolgende Angabe „#ffffff“ für weiß dient nur als Auffangregelung.

**Kommentiert [RP22]:** Opazität (Deckkraft) der Hügel-farbe. Sollte normalerweise 1 für 100 % sein.

**Kommentiert [RP23]:** Dies ist die Farb für den erweiterten Schatten bzw. Halbschatten, der sich hinter den Schraffen befindet. In diesem Beispiel hier ist er schwarz,...

**Kommentiert [RP24]:** ... aber damit man ihn nicht sieht (sondern nur die Schraffen), ist seine Opazität auf 0 gesetzt.

**Kommentiert [RP25]:** Farbe für die Beleuchtung der Hügel von Westen her. Sollten normalerweise weiß sein. Ist nur zu sehen, wenn die Hügel-farbe (s.o.) dunkler ist als weiß.

**Kommentiert [RP26]:** Strichstärke für die Schraffen. Wird in QGIS gemeinsam mit der Strichstärke für die übrigen Konturen eingestellt.

**Kommentiert [RP27]:** Konturfarbe für die Schraffen. Wird in QGIS gemeinsam mit der Farbe für die übrigen Konturen eingestellt.

**Kommentiert [RP28]:** Farbe des ganz dunklen Haupt-Schattens, hier schwarz.

**Kommentiert [RP29]:** Deckkraft für den Haupt-Schatten. Achtung: QGIS denkt, dies beziehe sich auf die Deckkraft der Hügel-farbe (weiß, s.o.)! Ich musste es für den Schatten zweckentfremden, weil die Parametrisierungsmöglichkeiten in QGIS derzeit offenbar noch defizitär sind.

**Kommentiert [RP30]:** Strichstärke für alle Konturen (einschl. Schraffen). Die anschließende „0.35“ dient nur als Auffangwert, wenn nichts anderes bestimmt wird.

**Kommentiert [RP31]:** Strichfarbe für alle Konturen (einschl. Schraffen). Das anschließende „#000000“ fü...



```

    }

    for old, new in replacements.items():
        text = re.sub(old, new, text)

    return text

# Funktion zum Finden des nächsten "alt##"-Verzeichnisnamens und dessen Erstellung
def create_backup_directory(base_path):
    existing_dirs = [d for d in os.listdir(base_path) if os.path.isdir(os.path.join(base_path, d)) and
d.startswith('alt')]
    highest_num = 0
    for d in existing_dirs:
        try:
            num = int(d[3:])
            highest_num = max(highest_num, num)
        except ValueError:
            continue # Falls das Verzeichnis nicht der erwarteten Benennung folgt, ignorieren

    # Nächsten Verzeichnisnamen festlegen und Verzeichnis erstellen
    next_dir_name = f'alt{highest_num + 1:02d}'
    next_dir_path = os.path.join(base_path, next_dir_name)
    os.makedirs(next_dir_path, exist_ok=True)
    return next_dir_path

# Funktion zum Kopieren der SVG-Dateien ins Sicherungsverzeichnis und Durchführen der Ersetzungen
def backup_and_update_svgs(directory, backup_directory):
    for i in range(1, 64): # Für MH01.svg bis MH63.svg
        file_name = f'MH{i:02d}.svg'
        file_path = os.path.join(directory, file_name)

        if os.path.exists(file_path):
            # Datei ins Sicherungsverzeichnis kopieren
            shutil.copy(file_path, backup_directory)

```

**Kommentiert [RP32]:** Dies und der folgende Code sorgen dafür, dass alle vorhandenen Hügeldateien in einem Unterverzeichnis namens „alto1“ (bzw. falls schon vorhanden, „alto2“ usw.) abgelegt wird.

```
# Datei öffnen, lesen und Ersetzungen durchführen
with open(file_path, 'r', encoding='utf-8') as file:
    content = file.read()
updated_content = replace_content(content)

# Geänderten Inhalt zurückschreiben
with open(file_path, 'w', encoding='utf-8') as file:
    file.write(updated_content)

# Hauptskript
def main():
    current_directory = os.getcwd()
    backup_directory = create_backup_directory(current_directory)
    backup_and_update_svgs(current_directory, backup_directory)

# Den Aufruf der Hauptfunktion aktivieren, um das Skript auszuführen
main()

# Dieses Skript enthält alle notwendigen Funktionen und Schritte, um die angeforderten Ersetzungen durchzuführen und
gleichzeitig eine Sicherungskopie der Originaldateien anzulegen.
```