**Dokumentation**

für den Dienstgebrauch

# Squad Maprotagenerator 3.0

**von**
**Timbow, Fletschoa, KappaKay**

SK Discord, 12.03.2023

# Inhaltsverzeichnis

**Abbildungsverzeichnis** **4**

**1 Einleitung** **5**
1.1 Problemdarstellung . . . . . 5
1.2 Ziel . . . . . 6

**2 Aufbau** **7**
2.1 Grundlegende Struktur . . . . . 7
2.2 Aufbau im Detail - Mode . . . . . 8
2.3 Aufbau im Detail - Map . . . . . 8
2.3.1 Map Charakteristiken . . . . . 9
2.3.2 Distanzweight . . . . . 10
2.3.3 Mapvoteweight . . . . . 10
2.3.4 Berechnung der Mapwahrscheinlichkeit . . . . . 11
2.3.5 Memory Kernel . . . . . 12
2.3.6 Auswahl der Map . . . . . 13
2.4 Aufbau im Detail - Layer . . . . . 13

**3 Optimizer** **15**
3.1 Funktionsweise . . . . . 15
3.2 Anwendung . . . . . 16
3.3 Temperaturprofil . . . . . 16
3.4 Startwerte . . . . . 16

**4 Ergebnisse** **17**
4.1 Bewertung des Systems . . . . . 17
4.1.1 Mapverteilung . . . . . 17
4.1.2 Mode/Modus Verteilung . . . . . 19
4.1.3 Varriation der Maps . . . . . 19
4.1.4 Patternbildung . . . . . 20
4.1.5 Map Wiederholung . . . . . 21
4.2 Grenzen des Systems . . . . . 22

**5 Zusammenfassung** **23**
5.1 Diskussion . . . . . 23
5.2 Ausblick . . . . . 23

Memory Colonel, der du bist in GooseBay, geheiligt werde dein Name.
Deine Rota komme.
Deine Locktime geschehe.
Unser täglich Squad gib uns heute.
Und vergib uns unser Minen legen, wie auch wir vergeben unseren Snipern
Und führe uns nicht in Versuchung, sondern erlöse uns von Tallil
Denn dein ist die Rota und Biome und die Abwechslung in Ewigkeit.
Amen

*- Das Maprota Gebet*

# Abbildungsverzeichnis


1 Oberflächlicher Ablauf des Generators. . . . . 7
2 Skizze Kugeloberfläche für drei verwendete Biome, die verwendeten Radien sind nicht in der Größenordnung die im Generator verwendet wird. Die schwarzen Punkte sind Maps, die gezogen wurden und noch im Memory Kernel liegen. Die grünen Punkte sind Maps, welche für die nächste Runde zur Verfügung stehen. Die roten Punkte sind Maps, welche gesperrt wurden aufgrund der Nähe zu einer bereits gezogenen Maps. . . . . 10
3 Die Weightfunktion $w_l(x) = f(x)$ für verschiedene Werte des ungewichteten Mittels $\mu$. Je weiter weg ein Layervote $x$ vom Mittelwert liegt, desto kleiner ist das Weight und desto sträker wird dieses Layer damit bestraft. 12
4 Die Sigmoidfunktion für mehrere Werte $a$ und $b$. Für $b = 0$ ist für alle Werte $a$ $f(0) = 1/2$. Der Parameter $b$ dient als Offset. . . . . 13
5 Erwartete Mapverteilung im Modus RAAS nach Layervotes vom 12.03.2023 18
6 Generierte Mapverteilung im Modus RAAS nach Layervotes vom 12.03.2023 (1.Mio. Layer Rota) . . . . . 18
7 Häufigkeiten des gleitenden Mittelwertes der Distanzen . . . . . 19
8 Pattern Auftrittswahrscheinlichkeiten ($1 = 100\%$) für die Länge $d = 3$. Es wurde die Häufigkeit für alle Patterns geplottet. Da es hier prinzipiell $22^3 = 10648$ verschiedene Patterns gibt wurde hier auf eine Achsenbeschriftung verzichtet. Jeder Wert entspricht einem spezifischen Pattern wie z.b. „Mutaha - Gorodok - Tallil“. Das Maximum wurde zu $p_{\text{mat}} = 0.0012$ bestimmt. . . . . 21
9 Häufigkeiten der Map Wiederholung . . . . . 22

# 1 Einleitung

Der gemeine Squad Spieler mag nach der Sammlung an einiger Erfahrung gemerkt haben, dass eine abwechslungsreiche Rotation von Layern (Maprota) die Qualität das Spieles deutlich verbessert. So fiel uns in der Vergangenheit auf, dass auf dem We ♡ Squad Discord sich immer wieder Spieler über die Maprota beschwert haben. Anfangs beschwerten wir uns auch, bis wir uns das Ziel gesetzt haben, ein Programm für die Generierung einer besseren Maprota zu schreiben, bevor wir uns wieder beschweren. Die Ziele, für das Projekt, waren schnell aufgestellt und das Problem scheint im ersten Moment einfach lösbar zu sein. Wir können euch nach 2 Monaten Entwicklungszeit aber sagen es ist definitiv nicht einfach! Es wäre einfacher gewesen sich weiterhin zu beschweren. Die Lösung, die wir für eine bessere Maprota gefunden haben, möchten wir hier vorstellen.

## 1.1 Problemdarstellung

Aus den historischen Debatten über die Maprota wurde versucht, die Hauptprobleme hervorzuheben und werden im folgenden erklärt.

Der Mapgenerator soll qualitativ hochwertige Maprotas generieren. Zur Klassifizierung werden wir nun zunächst Eckpunkte definieren welche die Qualität einer Rota bemessen sollen.

Zunächst sollte sich eine Map nicht zu stark wiederholen. Des Weiteren gab es in der Community bedenken, welche sich damit beschäftigen, dass nicht zu ähnliche Maps zu kurz hintereinander gespielt werden sollten. Ein Beispiel für letzteres wäre die Abfolge

$$\text{Sumari} \rightarrow \text{Logar Valley} \rightarrow \text{Fallujah}.$$

Hier würden direkt nacheinander folgend drei relativ ähnliche Maps gezogen werden. Der Charakter dieser Maps wird im wesentlichen über die Eigenschaften „Wüste“, „Stadt“, „Infanterie lastig/klein“ der Map definiert. Eine gute Rota sollte solche Map-Ketten vermeiden.

Ein weiterer wichtiger Punkt ist die Vermeidung von Mustern in der Rota. Das bedeutet, dass die generierten Rotas nicht deterministisch verteilt, sondern aus zufälligem Ziehen entstehen sollen.

Das seit einiger Zeit bestehende Layervote-System muss direkten Einfluss auf die Rota haben um die Verteilung den Wünschen der Community anzupassen.

## 1.2 Ziel

Zusammenfassend werden wir im folgenden die Qualität der Maprota an folgenden Eigenschaften messen:

- Ähnlichkeit der gezogenen Maps in kurzem Zeitraum
- Wiederholung der selben Map in kurzem Zeitraum
- Keine Muster/Nicht-deterministische Verteilung
- Layervotes müssen direkten Einfluss haben
- hintereinander gespielte Maps sollen so gewählt sein dass ein Spieler nicht mehrmals hintereinander die selbe Fraktion spielt

Der in diesem Dokument beschriebene Algorithmus soll alle fünf genannten Eigenschaften so gut wie möglich Erfüllen. Kurz gesagt ist das Ziel des Maprota Generators: „Ein probabilistisches System dessen globale Verteilung durch Mapvotes gegeben ist und lokal Wiederholung ähnlicher Maps vermeidet.“ Anders ausgedrückt:
Das System sollte global einer Verteilung folgen welche die Map/Layervotes wiederspiegeln und lokal eine gewisse Varianz unter den Mapcharakteristiken hervorruft.

Im weiteren Verlauf wird ein Algorithmus präsentiert, welcher alle oben genannten Aspekte so gut wie möglich abdeckt. Es sei aber darauf hingewiesen, dass nicht alle Punkte gleichzeitig perfekt umgesetzt werden können aufgrund kritischer Eigenschaften des Systems und der gesetzen Nebenbedingungen. Darauf wird im Folgenden noch näher eingegangen. Insbesondere das Hinzufügen weiterer Nebenbedingung machen eine weitere Abbildung des Systems zunehmend schwieriger.

# 2 Aufbau

Im Folgenden wird der Aufbau des Algorithmus näher beschrieben. Nach einem kurzem Überblick werden die einzelnen Subsysteme detailliert erklärt.

## 2.1 Grundlegende Struktur

Es soll eine Liste an Layern, anhand der zuvor definierten Ziele, generiert werden. Die oberflächliche Struktur, des Generators, ist im unteren Flussdiagramm dargestellt. Für eine gegebene Anzahl an Layern in einer Rota wird für jedes Layer folgende Routine ausgeführt:

- wähle einen Modus, gewichtet nach den Mode-Wahrscheinlichkeiten $w_g$ und dem Mode-Spacing $\Delta_g$
- für den gewählten Modus werden die validen Maps gefiltert
  - zunächst wird geprüft, welche Map im Modus repräsentiert ist
  - aus den vorhanden Maps wird nach dem Distanz-Vote-Weight $w_m$ gewichtet eine Map gezogen
- für die gezogene Map wird gewichtet nach dem Layerweight $w_l$ ein Layer gezogen
- die gezogenen Maps verbleiben für eine feste Maplänge im Memory-Kernel und sind damit für die nächsten Map Ziehungen nicht verfügbar

Im Folgenden werden die drei verschiedenen Schritte genauer erklärt und die verwendeten Weights definiert.

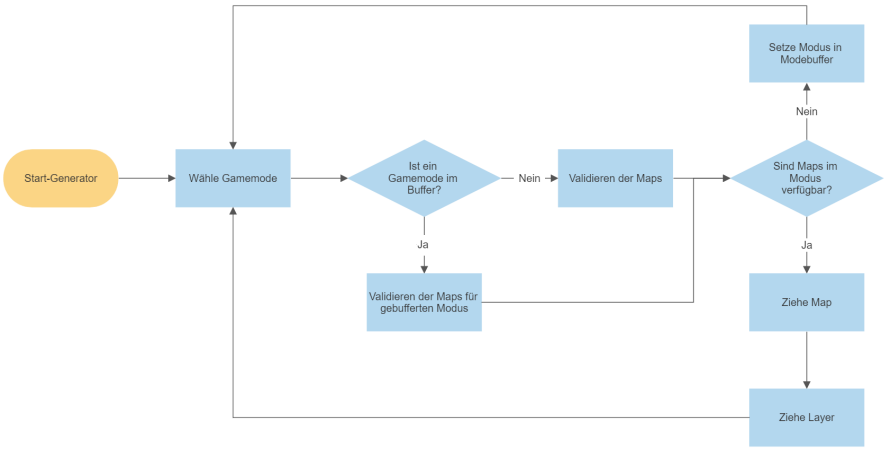


Abbildung 1: Oberflächlicher Ablauf des Generators.

## 2.2 Aufbau im Detail - Mode

Wie zuvor erwähnt gibt es zwei Faktoren, welche die Ziehung des Modus beeinflussen:

1. die Modeweights die zuvor gesetzt wurden $w_g$
2. das Modespacing $\Delta_g$

Des Weiteren erlaubt der Generator eine Gruppierung der Modes in sogenannte „Pools“. Es gibt immer einen sogenannten *main*-Pool. Ohne weitere Einstellungen ist jeder Modus, der gespielt werden soll, automatisch im Main-Pool enthalten. Neben diesem können noch weitere Pools definiert werden. In der momentanen Fassung existieren drei Pools:

- **main**: Der zuvor erwähnte Standard-Pool, beinhaltet *RAAS* und *AAS*
- **intermediates**: Beinhaltet die Modi *Invasion* und *TC*
- **reste**: Beinhaltet *Destruction* und *Insurgency*

Das **Modespacing** $\Delta_g \in 0, ..., N$ und maximal die Anzahl der Layer in einer Rota. Es sorgt nun dafür, dass für eine gegebene Anzahl an Runden nur der Main-Pool gezogen werden darf, wodurch eine „Mindestzeit“ zwischen z.b. zwei Invasion Layern garantiert wird. Sollte die Zeitspanne seit dem letzten nicht-main Modus größer sein als das Modespacing, so wird mit den Pool und Mode-weights gewichtet ein Gamemode gewählt. Hierbei wird erst der Pool ausgewählt und anschließend im Pool der Modus. Es ist zu beachten, dass dies dazu führen kann und wird, dass nicht direkt nach Ablauf des Modespacings ein andere Pool drankommen muss. Das Design wurde mit Absicht so gewählt, um repetitive Modes zu verhindern.

Das Modeweight $w_g$ setzt sich (für $\Delta_g = 0$) damit zusammen aus der Wahrscheinlichkeit den enthaltenen Pool zu ziehen und den Modus

$$w_g = \mathbb{P}(G = g|P = p) = \mathbb{P}(G = g)\mathbb{P}(P = p) \tag{1}$$

wobei hier $G$ und $P$ die Zufallsvariablen „Gamemode“ und „Pool“ darstellen sollen. Wenn $\Delta_g \geq 0$ ist muss der Memory Effekt in obiger Formel noch inkludiert werden, allerdings gibt diese eine gute erste Näherung. Durch das Modespacing ist die Wahrscheinlichkeit einen Modus zu ziehen ungleich größer als $w_g$. Dies muss bei der Wahl der Mode und Pool Wahrscheinlichkeiten beachtet werden.

Anschließend wird geprüft ob die Verfügbaren Maps den gewählten Modus enthalten. Sollte dies nicht der Fall sein so wird der Modus in einem Buffer gespeichert und es wird erneute ein Modus zufällig gewählt. Der Buffer wird bei der nächsten Gelgenheit abgearbeitet.

## 2.3 Aufbau im Detail - Map

Nachdem ein Mode gewählt wurde folgt das Auswählen einer Map. Für das Ziehen der Maps werden Mapweights verwendet welche im wesentlichen von den Layervotes und der „Ähnlichkeit“ der gespielten Maps abhängen. Zudem wird noch ein Memory

Kernel verwendet der sich die letzten $k \geq 0$ gespielten Maps merkt um damit ungewollte Mapwiederholungen zu vermeiden.

Das Weight errechnet sich als Produkt aus einem „Distanz-Weight“ und einem „Mapvote-Weight“.

$$w_m(m, d, v) = \frac{1}{\mathcal{N}} w_d(d, m) w_v(v, m) \tag{2}$$

mit $d$ der Distanz zur zuletzt gezogenen Map, $m$ der gewählten Map und $v$ die Summe aller Layervotes der Map im entsprechnden zuvor gewählten Modus. $\mathcal{N}$ ist die Norm des weights sodass $\sum_m w_m = 1$. $w_d(d, m)$ ist das sogenannte „Distanzweight “und $w_v(v, m)$ das interne Mapweight, welches von den Layervotes abhängt.

### 2.3.1 Map Charakteristiken

Um die verschiedenen Maps voneinander zu unterscheiden wird versucht bestimmende Charakteristiken der einzelnen Maps. Das heißt es werden bestimmte Eigenschaften der Map bewertet um diese untereinander zu vergleichen. Zum Beispiel kann man das „Setting“ bewerten wie „Wüste“ oder „Schnee“ und damit verhindern das zu oft Wüstenmaps hintereinander gespielt werden. Die Idee ist eine Metrik $d(m_1, m_2)$ einzuführen welche einen „Abstand“ zwischen zwei Maps $m_1$ und $m_2$ über die Map Charakteristiken misst. Zurzeit werden folgende Eigenschaften bewertet: Für jeder Map wird jeder Eigenschaft

| **Mapcharakteristik** |
|---|
| Mapgröße |
| Wald |
| Schnee |
| Wasser |
| Wüste |
| Grasland |
| Stadt |
| Berge |
| Felder |

Tabelle 1: Benutzte Mapcharakteristiken.

ein Wert zwischen 0 und 1 zugewiesen, wobei ersteres dafür steht, dass die Eigenschaft von der Map gar nicht erfüllt wird und letzteres bedeutet dass die Map diese Eigenschaft voll erfüllt. Die Einträge werden dann in einen Vektor $\vec{b} = (b_1, ..., b_N)^T$ der aus den $N$ werten besteht zugeordnet und dieser wird Anschließend normiert sodass $|\vec{b}| = 1$. Damit lässt sich jede Map als Punkt auf einem Teilstück einer $N$-Dimensionalen Kugel beschreiben. Dadurch kann man eine Distanz zwischen zwei Punkten auf besagtem Flächenstück definieren, welche gegeben ist durch

$$d(m_1, m_2) = 2\arccos(\vec{b_1} \cdot \vec{b_2}), \tag{3}$$

wobei $\vec{b_1}$ die Mapcharakteristiken der Map $m_1$ beschreibt und $\cdot$ das Standardskalarprodukt ist. Damit ist die Distanz zwischen zwei Maps rein durch den Winkel der beiden

Vektoren gegeben. Maps, die ähnlich sind, wie z.b. „Logar“ und „Sumari“, liegen nahe

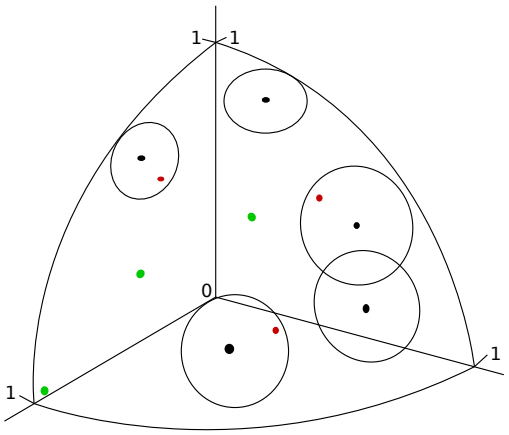


Abbildung 2: Skizze Kugeloberfläche für drei verwendete Biome, die verwendeten Radien sind nicht in der Größenordnung die im Generator verwendet wird. Die schwarzen Punkte sind Maps, die gezogen wurden und noch im Memory Kernel liegen. Die grünen Punkte sind Maps, welche für die nächste Runde zur Verfügung stehen. Die roten Punkte sind Maps, welche gesperrt wurden aufgrund der Nähe zu einer bereits gezogenen Maps.

beieinander sind aber weit entfernt von z.B. "Yehorivka".

### 2.3.2 Distanzweight

Das Distanzweight ist eine allgemeine stückweise stetige Funktion definiert durch

$$w_d : \mathbb{R}^+ \to \mathbb{R}^+, d \mapsto w_d(d), \tag{4}$$

und der Nebenbedingung $w_d(d) \overset{d\to 0}{\longrightarrow} 0$. In der momentanen Version ist die Funktion gegeben durch

$$w_d(d) = 1_{[0,d_{\min}]}(d) \tag{5}$$

mit $d_{\min} \geq 0$ als Mindestdistanz und $1_{[a,b]}(x)$ der Indikatorfunktion auf dem Intervall $[a, b]$. Sollte eine Map näher als $d_{\min}$ an einer zuvor gezogenen Map liegen, ist das Distanzweight und damit das Mapweight 0 und wird damit nicht gezogen.

### 2.3.3 Mapvoteweight

Das Mapvoteweight soll die Beliebtheit der Map repräsentieren. Es ist allerdings nicht die Mapwahrscheinlichkeit gemessen an den Votes sondern ein rein internes Weight.

Dies kommt daher, dass die Wahrscheinlichkeit eine Map zu ziehen durch den Memory Kernel und das Distanzweight stark nicht-linear vom Mapvoteweight abhängt und damit keine 1-zu-1 Relation gegeben ist. Die internen Weights müssen so gewählt sein, dass die Mapwahrscheinlichkeit zu der vorhergesagten Mapwahrscheinlichkeit passt. Leider ist dieses Problem nichtmehr geschlossen analytisch lösbar und benötigt numerische Lösungsmethoden. Näheres dazu im Optimizer Kapitel. Im allgemeinen ist bekannt dass das interne Weight von der Anzahl an „Nachbarn "von welchen diese gesperrt werden kann abhängt und von der gewollten Map-Auftrittswahrscheinlichkeit. Mit Nachbar einer Map, ist jede Map gemeint, die näher als $d_{min}$ an der Map liegt.

$$w_v(v) = \sum_{i,j=0}^{2} a_{ij} x^i y^j \tag{6}$$

wobei $x$ die Anzahl an verbundenen Maps ist und $y$ die Mapwahrscheinlichkeit errechnet aus den Layervotes. Die Koeffizienten $a_{ij} \in \mathbb{R}$ müssen vorerst numerisch bestimmt werden. Die obige Definition für $w_v(v)$ ist nur eine Näherung und wird als Startwert für den Numerischen optimierer verwendet.

### 2.3.4 Berechnung der Mapwahrscheinlichkeit

Um die internen Mapweights zu bestimmen, muss die Beliebtheit einer Map anhand der Layervotes errechnet werden. Dies erfolgt in zwei Schritten, getrennt für jeden einzelnen Modus.

- **Gewichtetes Mittel der Layervotes**
  Hierbei wird ein Mapvote als mittleres Layervote errechnet. Dafür wird ein gewichtetes arithmetisches Mittel verwendet damit „Ausreißer" die Mapbeliebtheit nicht zu stark beeinflussen. Der Mapvote ist gegeben durch

  $$\bar{v}_m = \sum_{l=1}^{N_l} \bar{w}_l v_l \tag{7}$$

  wobei $N_l$ die Anzahl an Layern, $v_l$ der Layervote-Wert eines Layers $l$ ist und das weight definiert ist als

  $$\bar{w}_l = \frac{w_l}{\sum_l w_l} \tag{8}$$

  mit

  $$w_l = \exp\left(-\left(v_l - \mu\right)^2\right) \tag{9}$$

  und $\mu$ der ungewichtete Mittelwert.

- **Mapweight und Wahrscheinlichkeit**
  Um eine Wahrscheinlichkeit zu bekommen wird zunächst der mittlere Vote auf das Intervall $[0, 1]$ abgebildet. Dies geschieht mittels einer Sigmoid Funktion, sodass

  $$S(w) = \frac{1}{1 + \exp\left(-a(w + b)\right)} \tag{10}$$

nun das Weight ist aus dem die Mapwahrscheinlichkeit errechnet wird welche nur von den Layervotes abhängen. $a$ und $b$ sind zwei freie Parameter welche die Steigung und den Offset beeinflussen. Die Funktion bildet große $w$ auf 1 ab und kleine $w$ auf 0. Werte die näher an der Null sind werden damit auf das Intervall zwischen den Randpunten abgebildet. Damit kann die Mapwahrscheinlichkeit durch die Votes kontinuierlich angepasst werden und eine Map „verschwindet“ erst nach mehreren negativen Votes aus der Rotation. Die Wahrscheinlichkeit, dass eine Map gezogen werden soll, ist dann gegeben durch

$$\mathbb{P}(M = m) = \frac{S(v_m)}{\sum_m S(v_m)} \tag{11}$$

und wird vor generation einer Rotaion zunächst berechnet. Dies erzeugt eine Verteilungsfunktion der Maps an welche die internen Mapweights angepasst werden müssen, bzw die Koeffizienten $a_{ij}$ müssen passend gewählt werden.

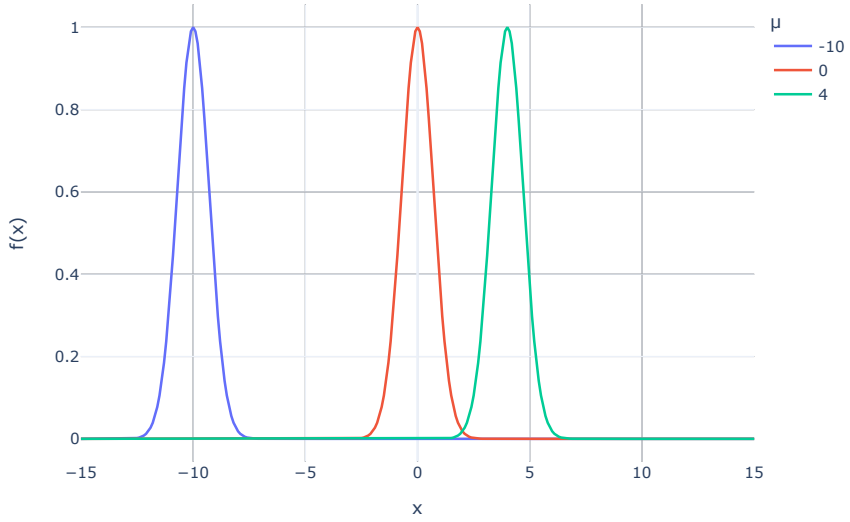


Abbildung 3: Die Weightfunktion $w_l(x) = f(x)$ für verschiedene Werte des ungewichteten Mittels $\mu$. Je weiter weg ein Layervote $x$ vom Mittelwert liegt, desto kleiner ist das Weight und desto sträker wird dieses Layer damit bestraft.

### 2.3.5 Memory Kernel

Um das Wiederholen von Maps zu vermeiden wird ein Memory Kernel („Gedächtnis“) verwendet. Das heißt, wenn ein Layer gezogen wurde, so wird die zum Layer zugeordnete Map in den Kernel gelegt. Das wird wiederholt, bis dieser mit einer Maximalen Anzahl

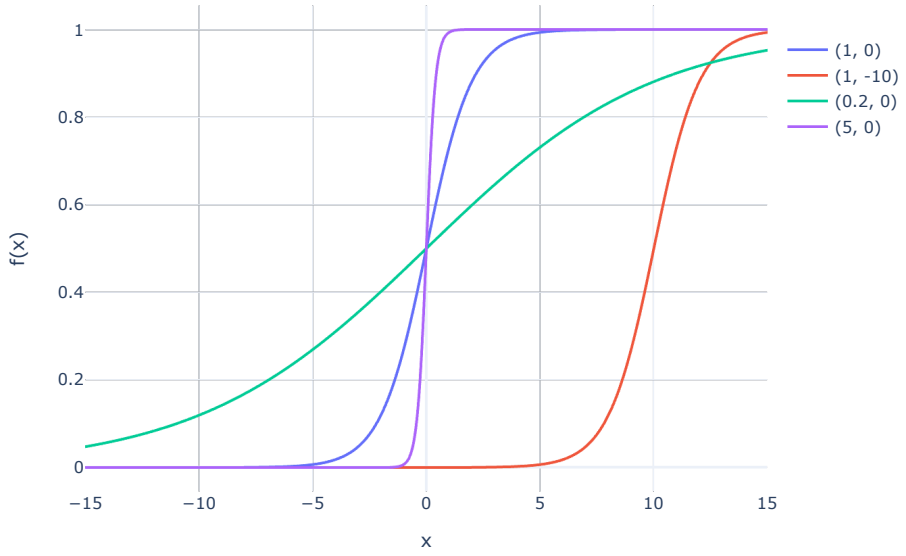


Abbildung 4: Die Sigmoidfunktion für mehrere Werte $a$ und $b$. Für $b = 0$ ist für alle Werte $a$ $f(0) = 1/2$. Der Parameter $b$ dient als Offset.

an Maps gefüllt ist. Dies wird dadurch implementiert, dass das Mapweight effektiv 0 ist für diese Map solange diese noch im Gedächtnis bleibt. Wenn nun in einem Modus eine Map gezogen wird, so darf diese nicht im Kernel liegen

### 2.3.6 Auswahl der Map

Nachdem nun alles notwendige definiert wurde wird kurz beschreiben wie eine Map gezogen wird.

- Nach Auswahl des Modus wird aus den vorhanden Maps zunächst das Distanzweight errechnet. Maps die zu Nahe an einer im Memory Kernel liegenden Map liegen können danach nichtmehr gezogen werden
- Aus den übrigen Maps wird gewichtet nach dem Mapweight eine Map gezogen
- Die ausgewählte Map wird in den Memory Kernel geschoben und sollte dieser voll sein, wird die älteste Map entfernt, wodurch ehemals gesperrte Maps wieder frei werden

## 2.4 Aufbau im Detail - Layer

Sobald eine Map ausgewählt wurde, wird das Layer gewichtet nach dem Layerweights $w_l$ gezogen. Diese hängen nur von den Layervotes ab, welche mit der vorher genannten

Sigmoidfunktion moduliert werden. Damit ist dann für $N_l$ Layer einer Map

$$w_l(j) = \frac{1}{\sum_{m=0}^{N_l} \frac{1}{[1+\exp(-a(v_m+b))]}} \frac{1}{1+\exp\left(-a(v_j+b)\right)} \tag{12}$$

mit $v_j$ die Mapvote-Zahl des Layers $j$ der gegebenen Map.

# 3 Optimizer

Wie zuvor erwähnt werden zum ziehen der einzelnen Maps interne Weights, definiert nach Gleichung 6, verwendet. Diese Weights sollen so gewählt sein dass beliebte Maps häufig genug gezogen werden, allerdings auch ein överfitting"durch zu starken Unterschieden in den vorhandenen Layer der Maps ausschließen. Dies soll alles unter Einhaltung der Map-Sperrzeiten erfolgen mittels statischer Weights. Um dieses nicht-triviale Optimierungsproblem unter Einhaltung der Nebenbedingungen zu lösen wird ein numerisches Verfahren verwendet welches den Namen „Simulated Annealing "trägt [1] Es wurde sich hier bewusst für ein stochastisches Optimierungsverfahren entschieden da erste Heuristik der Optimierung gezeigt hat dass es sich hier um ein hochgradid nicht-konvexes Optimierungsproblem handelt.

## 3.1 Funktionsweise

Zunächst wird ein optimizer benötigt. Dieser ist eine Funktion der internen Mapweights $w_i$ und der Mapwahrscheinlichkeiten und definiert durch

$$s(w_j) = \sqrt{\sum_i \left(p_i^{\text{fi}} - p_i^{\text{gen}}(w_j)\right)^2}. \tag{13}$$

Der zu betrachtende Optimizer ist als Maß der Abweichung der „Soll-Mapverteilung" und der Mapverteilung des Generators gedacht. Es seien $p_i^{\text{fi}}$ die Mapwahrscheinlichkeiten errechnet nach den Mapvotes und $p_i^{\text{gen}}$ die Mapwahrscheinlichkeiten nach dem Mapgenerator, sodass $\sum_i p_i^{\text{fi/gen}} = 1$. Da die Generator-Wahrscheinlichkeiten von den internen Mapweights abhängen ist $p_i^{\text{gen}} = p_i^{\text{gen}}(w_j)$ eine Funktion der Weight-Koeffizienten. Es gilt nun diese so zu Optimieren, dass der Optimizer

$$s(w_j) = \sqrt{\sum_i \left(p_i^{\text{fi}} - p_i^{\text{gen}}(w_j)\right)^2} \tag{14}$$

minimal wird. Das heißt wir versuchen die Gleichung

$$\min_{w_j} \quad s(w_j) = w_j^* \tag{15}$$

zu lösen. Hierbei ist aber im Allgemeinen $s_m(w_j^*) \neq 0$, da nicht angenommen werden kann, dass ein globales Minimum existiert.

Es wird nun ein Startwert $w_0 = (w_1, w_2, ..., w_N)$ an Weights ausgewählt, weitere Informationen dazu s.u.. Desweiteren wird dem System eine „Temperatur "$T > 0$ zugeordnet. Diese wird später sukzessiv reduziert. Der erste Schritt des Optimizers ist die Variierung des ersten Parameters um ein $\delta x > 0$. Dadurch erhält man einen neuen Parameter $w_j' \to w_j + \delta x$, welcher die generierten Mapwahrscheinlichkeiten ändert. Die neue Verteilung ersetzt damit die $p_i^{\text{gen}}$ von zuvor und ein neuer Optimizer Wert wird errechnet.

[1] Press, W. H.; Teukolsky, S. A.; Vetterling, W. T. & Flannery, B. P. 2007, Numerical Recipes 3rd Edition: The Art of Scientific Computing, Cambridge University Press.

Anschließend wird der neue Optimizerwert mit dem alten verglichen. Es folgt nun eine Entscheidung ob der neue s.g. State beibehalten wird oder doch wieder zum Originalen zurück gegangen wird. Der Shift in $w_j$ wird nun mit der Wahrscheinlichkeit

$$P(T) = \exp\left(-\Delta s(w_j)/T\right) \tag{16}$$

beibehalten wenn $\Delta s = s(w_j') - s(w_j)$ negativ ist, andernfalls wird der State immer aktzeptiert. Die Treshhold Wahrscheinlichkeit $P(T)$ reduziert sich für kleiner werdendes $T$ auf 0, was bedeutet dass jeder „schlechte "vorgeschlagene State wieder verworfen wird. Das heißt für $T \to 0$ wird der stochastische Algorithmus zu einem klassischen Hill-Climb Algorithmus. Diese Prozedur wird nun für jede Map sequenziell ausgeführt und nach durchlaufen aller Weights wird die Temperatur um einen schritt $\delta T > 0$ reduziert. Dieser Prozess wird solange wiederholt bis entweder eine Maximale Anzahl an Iterationen oder eine Minimaltemperatur $T_{\min} > 0$ erreicht wurde. Die berechneten Koeffizienten $w_j$ werden dann benutzt um die internen Mapweights zu berechnen.

## 3.2 Anwendung

Das Finden neuer Parameter durch den Optimizer wird nur bei veränderten Layervotes und veränderten Einstellungen benötigt. Daher wird der Optimizer auch nur bei solchen Veränderungen vor der Generierung automatisch aufgerufen. Für eine geringere Laufzeit wird die Optimizer parallel für jedem Modus ausgeführt.

## 3.3 Temperaturprofil

Prinzipiell gibt es mehrere Möglichkeiten wie die Temperatur reduziert werden soll, z.b. linear. Bei diesem Projekt wurde sich für ein exponentielles Profil entschieden mit $T(t) = T_0 \exp\left(-t * a\right)$ wobei $T_0$ die initial Temperatur ist, $t$ eine Zeitdifferenz zum ersten Optimierungsschritt und $a$ eine positive reele Zahl welche die Steigung beeinflusst.

## 3.4 Startwerte

Der Startwert $w_0$ ist hochgradig wichtig für ein Erfolgreiches Optimieren. Um einen guten Start zu finden wurden zunächst für jeden Modi einzeln Rotas mit zufälligen internen Weights $w_j$ generiert und die Mapverteilung gesammelt. Anschließend wurde das interne Weight als Funktion der Sollwahrscheinlichkeit $p^{\text{fi}}$ und $M$ der Anzahl an Maps im selben Cluster[2] mittels einem Polynom dritten Grades gefitted. Das heißt das Startweight jeder Map bestimmt sich als Funktion

$$(w_0)_j = \sum_{k=0}^{3} \sum_{i=0}^{3} \alpha_{ij} N^i (p^{\text{fi}})^j \tag{17}$$

wobei die Koeffizienten $\alpha_{ij}$ Numerisch durch einen Fit-Algorithmus bestimmt wurden wie er z.b. in Software wie *Mathematica* oder *Matlab* zu finden ist.

[2]Mit Cluster ist die Menge aller Maps gemeint welche eine Map Sperren können wenn diese gezogen werden sollten.

# 4 Ergebnisse

## 4.1 Bewertung des Systems

Um den entstandenen Maprotagenerator bewerten zu können und den Grad der Qualität festellen zu können, werden Metriken zur Hilfe genommen. Die Metriken sind für Squadmaprotas allgemeingültig und anhand dessen könnten sie miteinader verglichen werden. Es soll nicht unerwähnt bleiben, dass durch eine schlechte oder auch falsche Wahl der Einstellparameter das Maprotasystem leicht bis hin zu sehr stark beeinträchtigt werden kann. Genaueres dazu ist unter 4.2 nachzulesen. Daher ist bei dieser Bewertung zu berücksichtigen, dass von uns wohl überlegte Einstellparameter festgelegt wurden und als Referenz für Änderungen herangezogen werden sollten. Das Wählen passender Einstellparameter kann im User manual nachgelesen werden.

Es folgt die Auswertung der Maprota anhand vorgegebenen Einstellungswerten.

### 4.1.1 Mapverteilung

Die Mapverteilung wird von den Layervotes beeinflusst, dieses ist in Kapitel 2.3 nachzulesen. Diese Verteilung ist die Vorgabe für das System und es wird eine optimale Annäherung angestrebt. Da durch die Zielvorgaben die Verteilungsvorgabe nicht immer erreicht werden kann, tritt eine Abweichung in der Verteilung auf. Diese Abweichung wird hier als mittlere quadratische Abweichung (MSD) pro Modus angegeben. Für diese Auswertung wurden den Verteilungen genommen, die aus den Layervotes vom 19.09.2022 entstanden sind. Dabei ist zu beachten, dass der Modus TC zu diesem Zeitpunkt „Verbugged“ ist und daher in der Tabelle 2 nicht auftaucht.

| Modus | MSD |
|---|---|
| RAAS | 0.00442 |
| AAS | 0.00863 |
| Invasion | 0.00867 |
| Insurgency | 0.00924 |
| Destruction | 0.01906 |

Tabelle 2: mittlere quadratische Abweichung Mapverteilung

Um eine Vorstellung für die Zahlen zu Entwickeln wird im Folgenden die angestrebte und generierte Verteilung als Diagramm dargestellt (siehe Abbildung 5 und 6).

Bei der Betrachtung der Diagramme ist zu beachten, dass es sich hier nur um den Modus RAAS handelt. Beispielsweise ist die Karte AlBashrah hier deutlich unterrepräsentiert. Dies ist im Modus Invasion nicht der Fall, da die Layervotes dort für AlBashrah deutlich besser ausfallen. Zudem lässt sich erkennen, dass die Karte Yehorivka, Blackcoast und Gorodok nicht den angestrebten Verteilung erreichen. Dieses Phänomen wird im Abschnitt 4.2 näher behandelt.

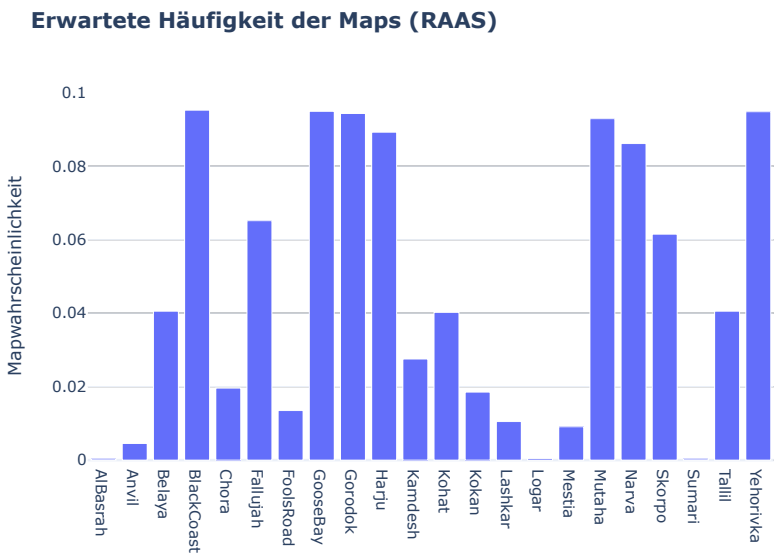


Abbildung 5: Erwartete Mapverteilung im Modus RAAS nach Layervotes vom 12.03.2023

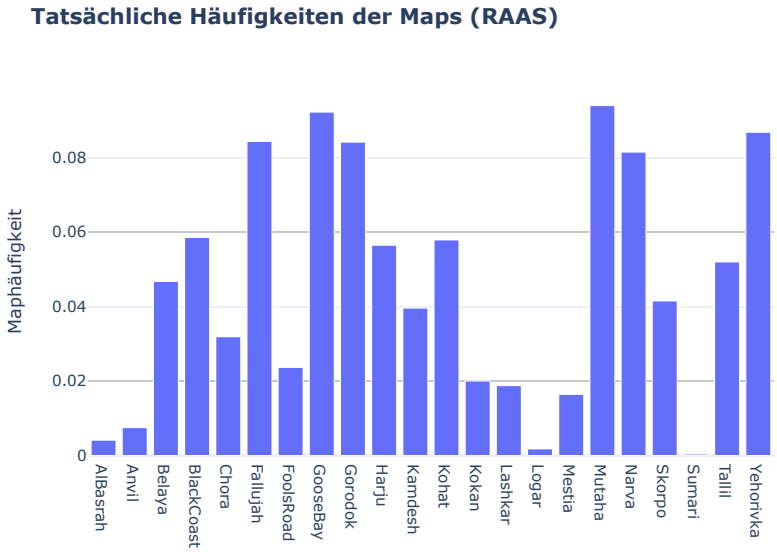


Abbildung 6: Generierte Mapverteilung im Modus RAAS nach Layervotes vom 12.03.2023 (1.Mio. Layer Rota)

### 4.1.2 Mode/Modus Verteilung

Wie bei den Mapverteilungen kann bei den Modusverteilungen die mittlere quadratische Abweichung als quantifizierendes Mitteln genommen werden. Bei den Modi ergibt sich eine MSD von 0.04514. Dieser Wert ist für die vorgesehenen Einstellparameter akzeptabel, da Modi die nicht RAAS oder AAS sind einen Mindestabstand haben. In diesem Falle ist dieser Abstand 4 Runden.

### 4.1.3 Varriation der Maps

Für den die Messbarkeit, der Differenz aufeinander folgende Maps, kann das arithmetische Mittel der Distanzen auf der Hyperfläche genutzt werden. Zudem ist es noch sinnvoll sich den gleitenden Mittelwert der Distanzen zu betrachten.
Das arithmetische Mittel der Distanzen beträgt $d_m = 1.08946$
Die Betrachtung des gleitenden Mittelwertes ergibt sich für eine Mittelwertbreite von 5 und einer Rota mit 100000 Layern eine Verteilung die auf Abbildung 7 zu sehen ist.

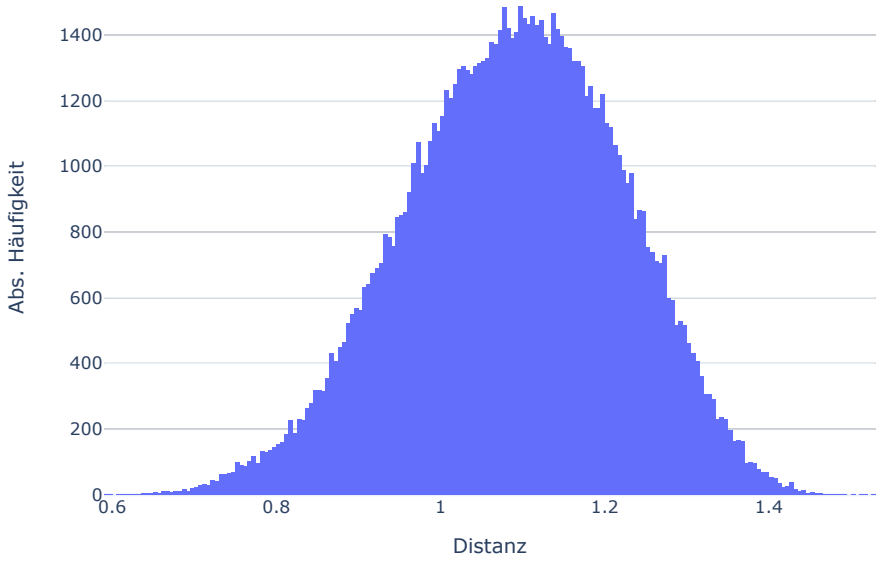


Abbildung 7: Häufigkeiten des gleitenden Mittelwertes der Distanzen

Die auf Abbildung 7 zusehende Normalverteilung zeigt das sich die höchste Häufigkeit zwischen dem eingestellten Minimum (0.4) und dem durch die Kugel gegebenen Maximum ($\pi/2$) befindet. Die Lage der Normalverteilung zeigt, dass es ein gutes Maß an

Diversität gibt. Zudem ist die Verteilung nicht nahe am Maximalrand welches eine Patternbildung begünstigen würde.

### 4.1.4 Patternbildung

Um das Auftreten von Patterns in der Rota zu untersuchen wird eine hinreichend große Rota-Historie generiert und anschließend die Häufigkeit aller Patterns betrachtet. Hierbei unterscheidet man die Länge $d$ eines Patterns welche durch die Anzahl an Maps innerhalb einer Mapabfolge definiert ist. Das heißt im allgemeinen wird es in jeder Rota $23^d$ verschiedene Patterns geben können bei 23 Maps. Der Ablauf der Patternbildungs-Prüfung ist wiefolg:

- generiere hinreichend viele Tagesrotas mit 30 Maps, ca. 250000 Rotas sind hier ausreichend
- für eine feste Länge $d$ suche alle auftretenden Patterns in den Rotas und summiere die Häufigkeiten
- Teile die Häufigkeiten für festes $d$ durch die Anzahl an gefunden Patterns
- wiederhole für jedes $d$

Es sei hier angemerkt dass die Pattern-Auftrittswahrscheinlichkeit $p_{\text{pat}} \overset{d\to\infty}{\longrightarrow} 0$ und im Regelfall $d \leq 5$ ausreichend ist.

**Definition Patternbildende Rota:** Ein Rota Generator wird als „Patternbildend "klassifiziert wenn mindestens ein Pattern innerhalb eines Zeitraums von $T = 30$ Tagen mit einer Wahrscheinlichkeit von $p_0 \geq 5\%$ mindestens zweimal auftreten wird. Die Wahrscheinlichkeit für das Auftreten dieses Events $X_0$ ist gegeben durch

$$\mathbb{P}(X_0) = 1 - Np_{\text{pat}}(1 - p_{\text{pat}})^{N-1} - (1 - p_{\text{pat}})^N \tag{18}$$

wobei $p_{\text{pat}}$ die Auftrittswahrscheinlichkeit eines Patterns ist und $N$ die Anzahl an gespielten Tagesrotas. Das bedeutet dass eine Maprota als „patternbildend"klassifiziert ist wenn mit mehr als 5% Wahrscheinlichkeit mindestens ein Pattern doppelt vorkommen kann innerhalb eines Monats. Zusammengefasst wird folgendes betrachtet

- berechne für festes $d$ die Auftrittswahrscheinlichkeiten $p_{\text{pat}}$ aller Patterns
- berechne $\mathbb{P}(X_0)$ das Maximale $p_{\text{pat}}$
- wenn $\mathbb{P}(X_0)$ unter dem Schwellwert von 5% liegt gibt es keine kritische Bildung
- wenn $\mathbb{P}(X_0)$ über dem Schwellwert liegt gibt es mindestens ein Pattern was häufiger auftritt

Im letzten Fall muss anschließend genauer untersucht werden wo die Patternbildung herkommt, wieviele Patterns betroffen sind und ob es Problematisch ist für die Rota. Die Abschätzung setzt ein Idealisiertes System vorraus und ist damit nur als obere

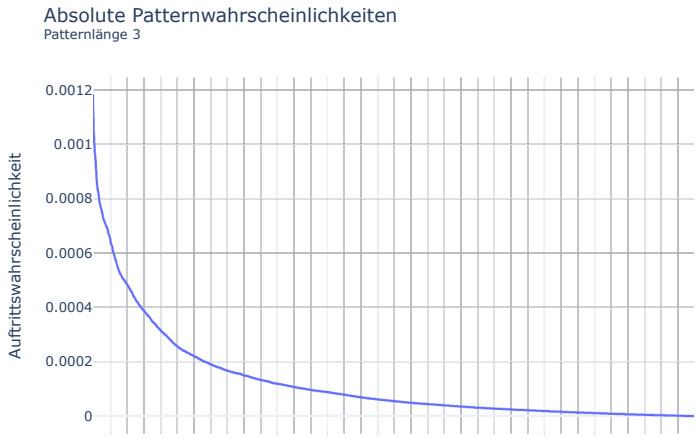


Abbildung 8: Pattern Auftrittswahrscheinlichkeiten ($1 = 100\%$) für die Länge $d = 3$. Es wurde die Häufigkeit für alle Patterns geplottet. Da es hier prinzipiell $22^3 = 10648$ verschiedene Patterns gibt wurde hier auf eine Achsenbeschriftung verzichtet. Jeder Wert entspricht einem spezifischen Pattern wie z.b. „Mutaha - Gorodok - Tallil“. Das Maximum wurde zu $p_{\text{mat}} = 0.0012$ bestimmt.

Schranke zu verstehen. Das heißt selbst wenn die Auswertung Patterns aufweist kann es trotzdem sein dass diese nicht im Live-System auffällt da Täglich weitaus weniger Layer gespielt werden als in der Theorie angenommen. Für die jetzige Implementierung sieht man die Wahrscheinlichkeit zur Auffindung von Patterns in Abbildung 8. Die maximale Wahrscheinlichkeit ist bei Patternlänge drei gegeben durch $p_{\text{pat}} = 0.00118$. Damit ist die Threshold Wahrscheinlichkeit gegeben mit $\mathbb{P}(X_0) = 0.000592 = 0.059\%$ was deutlich unter der kritischen Wahrscheinlichkeit von $5\%$ liegt.

### 4.1.5 Map Wiederholung

Das nächste und hier letzte benutzte Mittel, um eine Maprota zu bewerten ist, nach wie vielen Runden sich eine Map wiederholt. Hierfür wurde eine Histogramm aus einer 100000 Layer Rota erstellt. Die Abbildung 9 zeigt die Häufigkeit einer Map Wiederholung. Es ergibt sich ein Minimum von 3 Runden bevor sich eine Map wiederholen kann. Am Häufigsten ist jedoch eine Map Wiederholung nach 6 bis 9 Runden. Dabei muss bedacht werden, das Squad aktuell (09.2022) 22 spielbare Maps beinhaltet. Daher ist dieses ein gutes Wiederholverhalten.

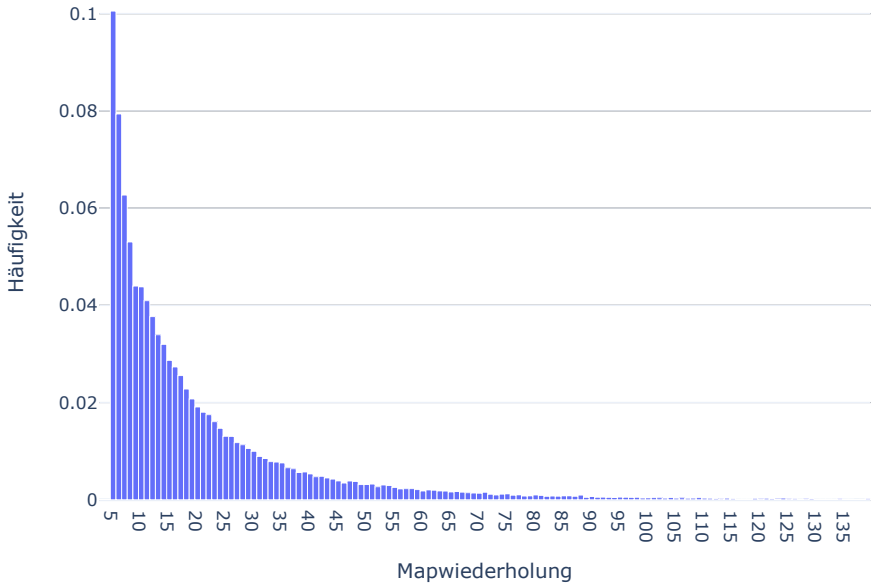


Abbildung 9: Häufigkeiten der Map Wiederholung

## 4.2 Grenzen des Systems

Um diese Sektion am besten nachzuvollziehen zu können, wird empfohlen, erneut einen Blick in das Kapitel 1.2 Ziele des Systems zu werfen. Es wird eine qualitativ hochwertige Maprotation gefordert, die zum Einen der Voteverteilung folgen soll und zum Anderen in der Map-Reihenfolge einige gewisse Diversität garantieren soll. Bei genauerer Überlegung ist das bereits ein Widerspruch in sich. Angenommen eine einzelne Map hat unendlich viele Stimmen und die Maprota folgt strikt den Votes. Es würde darin Resultieren, dass nur noch diese eine Map vorkommen dürfte. Diese, von der Maprotation angenommenen Verteilung, bildet aber einen Konflikt mit dem Ziel, dass die Maps eine gewisse Diversität bieten sollen. Daher sind an der Stelle die Möglichkeiten das Systems beschränkt und die Voteverteilung kann nicht immer voll in einer generierten Rotation abgebildet werden. Maps, die sehr viele Upvotes erhalten, können nur so oft drankommen, wie es Distanzweight $w_d$ und der Memory Kernel zulässt. Dieser Aspekt des Systems muss aber nicht als negativer Punkt aufgefasst werden, denn niemand möchte dauerhaft nur eine einzige Map spielen (solange es nicht GooseBay ist). Dieses „Feature“ wirkt damit aktiv gegen die Befürchtung, welche im Vorfeld angesprochen wurde, dass nur noch sehr beliebte Maps wie Yehorivka und Gorodok drankommen. Um trotzdem das Optimum zwischen vorgegebener Verteilung und Diversität der Maps zu garantieren wird ein Optimizer eingesetzt.

# 5 Zusammenfassung

## 5.1 Diskussion

In dieser Dokumentation wurde ein neuer Mapgenerator dargestellt, welcher für mehr Varianz aber auch gleichzeitig gute Einhaltung der Mapbeliebtheiten einhält. Dafür wurden zunächst Ziele definiert, welche eine Rota erfüllen sollte. Dazu zählen Vermeiden von Wiederholungen ähnlicher Maps in kurzem Zeitraum und globale Verteilung welche Mapvotes repräsentiert. Der Generator ist als Schicht-Modell aufgebaut, welcher zunächst den Modus, die Map und anschließend ein Layer zieht. Durch Einführung der Mapcharakteristiken können Maps auf Ähnlichkeit miteinander verglichen werden. Gezogene Maps werden durch einen Memory Kernel für kurze Zeit gesperrt um ungewollte Mapwiederholungen zu vermeiden.

Wie gezeigt wurde konnte mit dieser Implementierung die gewünschte Mapverteilung nach Mapvotes annähernd erreicht werden, jedoch kann, aufgrund des Sperrens der Maps, die gewünschte Mapverteilung nie perfekt widerspiegeln werden. Des Weiteren konnte gezeigt werden, dass die Entfernungen hintereinander gezogener Maps im Mittel die halbe Verfügbare Strecke benutzt und eine Varianz aufweist die nicht zu stark lokalisiert ist, sodass es zu Patternbildung kommen könnte. Letzteres wurde ebenfalls überprüft und es zeigt sich, dass die Häufigkeit auftretender Patterns nicht abhängig von der Patternlänge zu sein scheint, was ein Indiez dafür ist dass es nicht zu starker Patternbildung kommen kann.

Alles in allem wurden alle Ziele die zuvor formuliert wurden erreicht und der Generator zeigt keine Abnormalitäten oder statistische Probleme auf.

## 5.2 Ausblick

Der hier präsentierte Maprota Generator bietet einen guten Kompromiss zwischen Varianz und Beliebtheit der Maps. Allerdings gibt es natürlich immer Punkte, die noch verbessert oder geändert werden könnten. Es besteht die Möglichkeit, das Distanzweight durch eine stetige Funktion zu ersetzen. Zum Beispiel könnte beim Ziehen einer Map dem Punkt auf dem Kugelstück eine Temperatur $T(t) > 0$ zugeordnet werden, welche mit jeder weiteren gezogenen Map stückweise abnimmt. Dadurch könnte ein Wärmefluss auf der Kugel definiert werden und jeder Punkt erhält eine Temperatur kleiner als die des getroffenen Punktes. Daraus kann dann ein Weight bestimmt werden, welches verschwindet für $T(t = 0)$ und größer wird je „kälter“ ein Punkt ist. Dadurch wird eine stärkere Varianz zwischen den Maps erlaubt, läuft aber auch Gefahr, dass eine ähnliche Map theoretisch gezogen werden könnte, abhängig von der verwendeten Verteilung.

Eine weitere Verbesserung wäre das Aufteilen der Mapcharakteristiken in „Setting“ und „Gameplay“ Charakter der Map. So kann zum Beispiel mit dem Setting bewertet werden, ob es sich um „Wüste“oder „Schnee“ handelt und mit dem Gameplay Charakter ob die Map offen ist, Fahrzeug- oder Infanterielastig ist usw. Dies würde zur Verwendung von zwei Kugelflächenstücken führen aus denen dann ein gemeinsames Distanzweight errechnet wird. Das Problem hierbei könnte darin bestehen, dass Squad eigentlich nur drei Settings („Wüste“, „Schnee“, „Gemäßigt“) bietet und durch die asymmetrische

Mapverteilung, auf den Settings, es zu Problemen in der endgültigen Verteilung kommen könnte.

Zu guter Letzt wäre noch die Möglichkeit das ganze System so umzubauen, dass die Map-Schicht im Generator komplett auf die Layer-Schicht abbildet wird. Dadurch muss kein Mapvote mehr berechnet werden und es würde die Möglichkeit bestehen den ganzen Generator auf Layervotes basierend arbeiten zu lassen. Da die internen Weights allerdings numerisch gelöst werden müssen sorgt dies für eine extrem große Laufzeit des Optimizers. Insbesondere ist auch nicht sicher ob das interne Weight immer noch einer einfachen einer Polynom-Näherung folgen kann und wie viele Abhängigkeiten an Generator-Parameter wie „Anzahl Mapnachbarn“ es geben wird.