

# BACHELORARBEIT

## Abbildung von Learning-Algorithmen-Modellen in deren Reward-Funktion

David Müller

Entwurf vom 15. März 2021





# BACHELORARBEIT

## Abbildung von Learning-Algorithmen-Modellen in deren Reward-Funktion

David Müller

Aufgabensteller: Prof. Dr. Claudia Linnhoff-Popien

Betreuer: Thomas Gabor  
Thomy Phan

Abgabetermin: 1. Januar 2099





Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 1. Januar 2099

.....  
*(Unterschrift des Kandidaten)*



## **Abstract**

Für ML-Algorithmen gibt es eine Vielzahl von Modellen und Strategien, die genutzt werden können, um das Lernverhalten des Agenten zu kontrollieren und damit schlussendlich dessen Resultate zu verbessern. Bereits eine simple Erweiterung wie das Lernen auf einer Epsilon-Greedy-Policy fügt so schon neue Komponenten zum Lernalgorithmus hinzu. Wir untersuchen, inwieweit sich derartige Erweiterungen allein durch die Wahl der Reward-Funktion abbilden lassen und welche Auswirkungen dies auf das Lernverhalten sowie die Resultate des Agenten hat. Außerdem wird in diesem Zuge analysiert, welches eigentliche Ziel durch so eine Reward-Funktion umgesetzt wird. Für eine anschauliche Darstellung wird ein Landschaftsnavigationsproblem betrachtet, in dem der Agent in einem zufällig generierten Terrain den höchsten Gipfel finden soll.



# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
<b>2 Navigations-Problem</b>	<b>3</b>
2.1 Das Environment . . . . .	3
2.2 TODO Q-Table . . . . .	4
<b>3 Luna-Lander</b>	<b>7</b>
<b>Abbildungsverzeichnis</b>	<b>9</b>
<b>Tabellenverzeichnis</b>	<b>11</b>
<b>Listings</b>	<b>13</b>
<b>Literaturverzeichnis</b>	<b>15</b>



# **1 Einleitung**



## 2 Navigations-Problem

Wir betrachten zunächst ein Landschaftsnavigationsproblem. Der Agent soll in einer zufällig generierten Landschaft unterschiedliche Aufgaben lösen.

### 2.1 Das Environment

Das Environment für diese Experimentreihe bildet eine Gebirgslandschaft, über der ein Raster liegt, worauf sich der Agent bewegen soll. Hierbei soll jeder Punkt auf dem Raster eine Höhe besitzen. Außerdem soll die Landschaft zufällig generiert werden können.

Die simpelste Lösung hierfür wäre wohl, ein zweidimensionales Array mit zufälligen Zahlen zu füllen. Auf diese Weise erhält man ein für jede Koordinate eine zufällige Höhe. Wir wollen allerdings eine Landschaft erstellen, die organisch und natürlich aussieht.

**Perlin Noise** Um dieses Ziel zu erreichen verwenden wir *Perlin Noise*. Hierbei handelt es sich um eine Rauschfunktion, mit der sich sehr natürlich wirkende Texturen zufällig generieren lassen. Abbildung 2.1 zeigt eine simple Darstellung von zweidimensionaler Perlin Noise, bei der die generierten Werte über Farbwerten von Schwarz bis Weiß abgebildet werden.

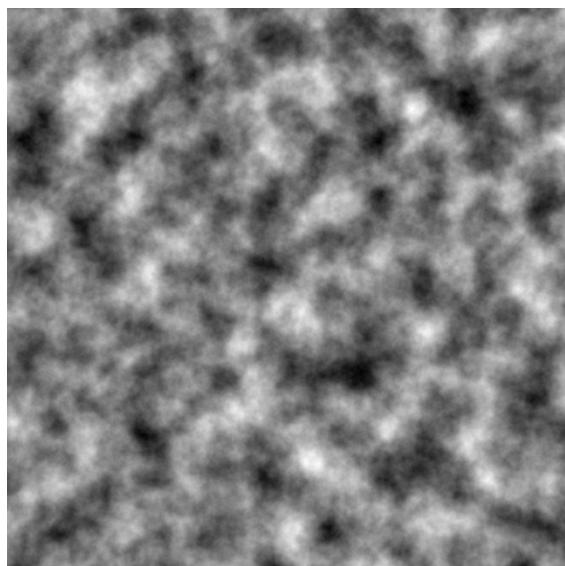


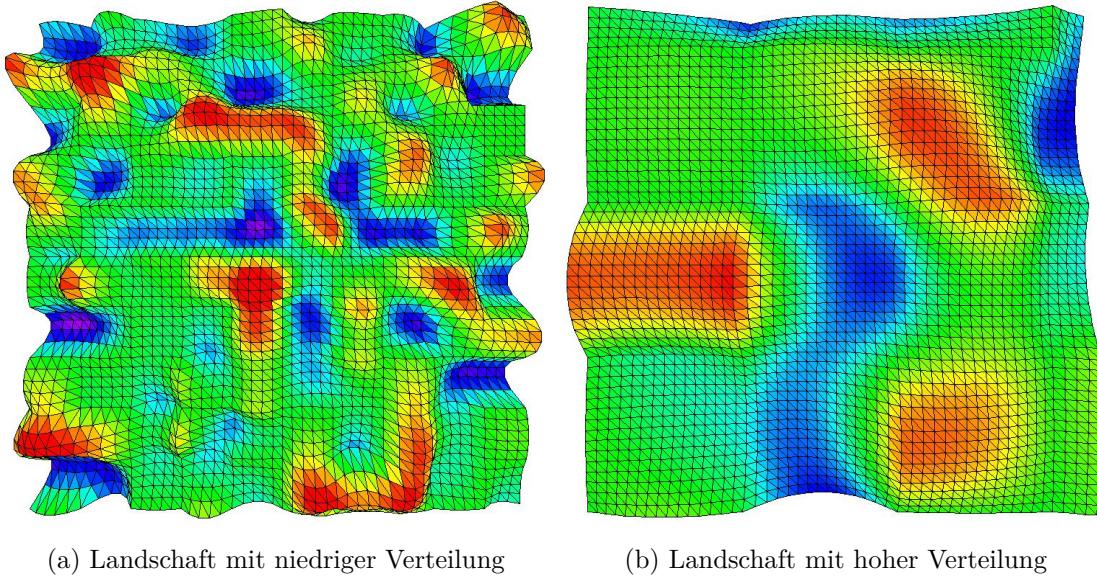
Abbildung 2.1: Visualisierung von zweidimensionaler Perlin Noise

Quelle: [https://miro.medium.com/max/2400/1\\*vs239SecVBaB4HvLsZ805Q.png](https://miro.medium.com/max/2400/1*vs239SecVBaB4HvLsZ805Q.png)

Perlin Noise ist nach [Par15] ein fundamentaler Algorithmus in der prozeduralen Generierung von Terrain und somit optimal geeignet, um unsere Umgebung zu erstellen.

## 2 Navigations-Problem

Wir verwenden eine modifizierte Implementierung von TODO, um ein zweidimensionales Array mit zufälligen Werten zwischen -1 und 1 zu erhalten, welche wir mit einer beliebigen Höhe multiplizieren können. Je nachdem, wie stark man in die Rauschfunktion „hereinzoomt“ erhält man unterschiedliche Verteilungen der Landschaft, wie man in Abbildung 2.2 erkennen kann.



(a) Landschaft mit niedriger Verteilung

(b) Landschaft mit hoher Verteilung

Abbildung 2.2: Mittels Perlin Noise zufällig generierte Landschaften

Wir werden nicht näher auf die Details der Funktion eingehen, da dies nicht Kern dieser Arbeit ist. Für weitere Ausführungen diesbezüglich verweisen wir auf [Arc11].

Für die Visualisierung der Landschaft benutzen wir eine abgewandelte Form des Codes von TODO. Zur besseren Differenzierung werden Berge und Täler zusätzlich zur perspektivischen Unterscheidung rot bzw. blau dargestellt.

Wir besitzen nun die Möglichkeit, eine zufällige Landschaft zu generieren und diese visuell darzustellen. Um bei allen Experimenten die gleichen Voraussetzungen zu gewährleisten, werden wir im folgenden den mittels der eben beschriebenen Methode zufällig generierten Terrain benutzen, der in Abbildung 2.4 zu sehen ist. Der höchste Punkt befindet sich bei dieser Landschaft auf dem Berg ganz oben in der Mitte.

## 2.2 TODO Q-Table

Um nun die erzeugte Landschaft zu testen, werden wir zunächst einen simplen Reinforcement Learning Agenten implementieren, welcher auf dem Terrain operiert.

422.52348pt

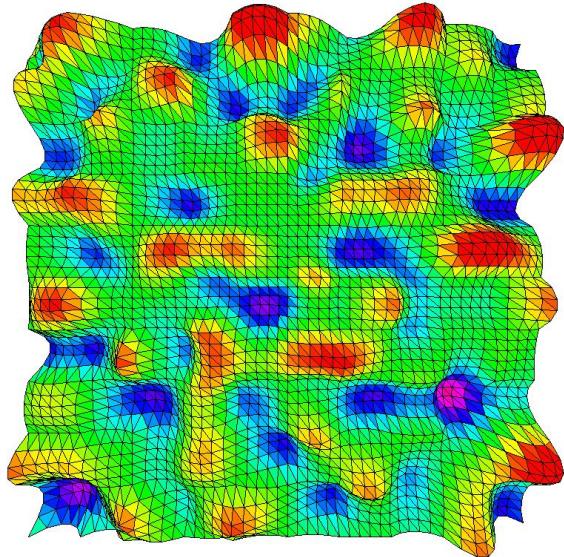


Abbildung 2.3: Visualisierung von zweidimensionaler Perlin Noise

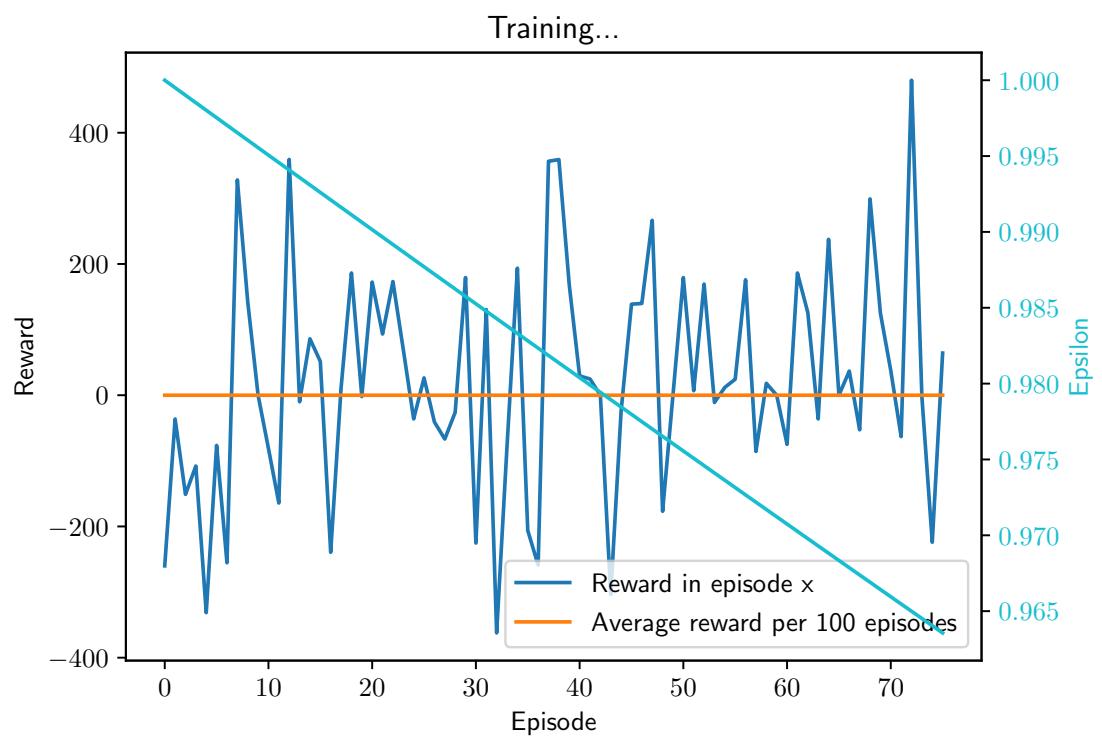


Abbildung 2.4: Visualisierung von zweidimensionaler Perlin Noise



### **3 Luna-Lander**



# **Abbildungsverzeichnis**

2.1	Visualisierung von zweidimensionaler Perlin Noise . . . . .	3
2.2	Mittels Perlin Noise zufällig generierte Landschaften . . . . .	4
2.3	Visualisierung von zweidimensionaler Perlin Noise . . . . .	5
2.4	Visualisierung von zweidimensionaler Perlin Noise . . . . .	5



# **Tabellenverzeichnis**



# Listings



# Literaturverzeichnis

- [Arc11] ARCHER, TRAVIS: *Procedurally generating terrain*. In: *44th annual midwest instruction and computing symposium, Duluth*, Seiten 378–393, 2011.
- [Par15] PARBERRY, IAN: *Modeling real-world terrain with exponentially distributed noise*. Journal of Computer Graphics Techniques, 4(2):1–9, 2015.