Reinforcement Learning

Ausarbeitung zum Fortgeschrittenen-Praktikum

Betreuer:	Paul A. Monderkamp	
Studierende:	Klett, Hanna	2951319
	Kocks, Steffen	2964775

vorgelegt am: 8. Juni 2023

Inhaltsverzeichnis

1 Einleitung **1**

2 Theoretische Grundlagen **1**
2.1 random-walk 1
2.2 $\mathcal{Q}$-learning 1

3 Aufgaben **3**
3.1 Mittleres Verschiebungsquadrat 3
3.2 Implementation des Lernalgorithmus 4
3.3 Wahl der Hyperparameter 6
3.3.1 Discount factor γ 6
3.3.2 Learning rate α 7
3.4 Stochastisches Hindernis 10

1 Einleitung

Das diesem Versuchsprotokoll zugrunde liegende Lernprojekt thematisiert die Grundlagen des *reinforcement learning* am Beispiel des $\mathcal{Q}$-learning Algorithmus. Betrachtet wird dazu ein eindimensionaler *random-walk* mit Diffusion. Darüber hinaus werden die grundlegenden Konzepte des Algorithmus bezüglich der Funktionsweise verschiedener Parameter vermittelt.

Implementiert ist der Algorithmus mittels objektorientierter Programmierung in `python3`.
(Der Quellcode ist dem Versuchsprotokoll angehangen.)

2 Theoretische Grundlagen

2.1 random-walk

Aus der statistischen Mechanik ist die kontinuierliche Form der Wahrscheinlichkeitsverteilung eines *random-walk* bereits bekannt. Bei gleicher Wahrscheinlichkeit der Schritte handelt es sich um eine Gaußverteilung

$$\mathcal{P}(x,t) = \frac{1}{\sqrt{4Dt}} \exp\left(-\frac{x^2}{4Dt}\right)$$

mit der **Diffusionskonstanten**

$$\boxed{D := \frac{a^2}{2\tau}},$$

wobei τ die mittlere Zeit zwischen zwei Schritten und $a = |\Delta x|$ die Schrittweite bezeichnet.

2.2 $\mathcal{Q}$-learning

Beim *reinforcement learning* liegen zu Beginn keine „Trainingsdaten“ für den Agenten (das ausführende Programm) bereit. Der Agent führt die nächste Aktion ausschließlich auf Basis des Feedbacks der vorausgegangenen Aktion aus. Diese „Erfahrungen“ bzw. „Erinnerungen“ werden in einer sog. *Learning-Matrix* $\mathcal{Q}$ gespeichert.

Damit der Agent eine effektive Strategie entwickeln kann, bearbeitet er die zu lösende Aufgabe im Laufe der Simulation wiederholt, bis er seine Strategie optimiert hat und somit als trainiert gilt. Eine solche Wiederholung wird als Episode bezeichnet. Mit einer Wahrscheinlichkeit ε wählt der Agent eine zufällige Aktion, ansonsten entscheidet er auf Basis der Einträge der Matrix $\mathcal{Q}$.

Im vorliegenden Beispiel befindet der Agent sich an einer Startposition auf einem eindimensionalen, diskreten Pfad. Die Aufgabe besteht darin, eine Zielposition zu erreichen und auf dieser zu verweilen. Die Zustände sowie mögliche Aktionen sind völlig diskret, d.h. es besteht eine feste Anzahl maximal möglicher Zustände sowie ein Set an möglichen Aktionen.

Im $\mathcal{Q}$-Learning wird die *Learning-Matrix* als Entscheidungsbasis verwendet:

$$\text{Aktion im Zustand } i \quad = \arg\max_j(\mathcal{Q}_{ij})$$

Indem diese Matrix mit dem Verlauf der Episoden ständig auf Basis der vorausgegangenen Aktionen sowie Zustände aktualisiert wird, ist es dem Agenten schließlich möglich, die Zielposition auf dem schnellstmöglichen Weg zu erreichen.

Die Aktualisierung der *Learning-Matrix* folgt folgender Relation:

$$\boxed{\mathcal{Q}_{ij}^{\text{neu}} = \mathcal{Q}_{ij}^{\text{alt}} + \alpha\left(\mathcal{R} + \gamma \max_{k}(\mathcal{Q}_{i'k}) - \mathcal{Q}_{ij}^{\text{alt}}\right)} \tag{1}$$

Die j Spalten von $\mathcal{Q}$ indizieren die möglichen bzw. vergangenen Aktionen. Der Index i bezeichnet den Zustand, in dem sich der Agent vor Durchführung der letzten Aktion befand, i' steht dementsprechend für den Zustand nach Ausführung einer Aktion. Der Maximums-Term bewertet die Aktionen im Zustand i', sodass die Information über eine zielführende Aktion mit jeder Episode um eine Stelle weiter propagiert. Der Agent lernt somit anhand der Position eines Ziels, ohne eine Information über das Vorhandensein eines Ziels zu haben.

Die weiteren Parameter sind:

- *learning rate* α:

  Dieser Hyperparameter des Machine Learning beeinflusst, mit welchem „Gewicht“ neu gewonnene Informationen die alten Informationen überschreiben. Beim $\mathcal{Q}$-Learning bestimmt α, wie stark die $\mathcal{Q}$-Matrix aktualisiert wird. Es gilt $0 < \alpha \leq 1$.

- *reward* $\mathcal{R}$:

  Der Parameter $\mathcal{R}$ beschreibt, welche maximale Belohnung für einen optimalen Schritt vergeben wird. In diesem Beispiel ist nur dann $\mathcal{R} \neq 0$, wenn der Agent im Zielzustand verweilt.

- *discount factor* γ:

  Dieser Hyperparamter bestimmt die Gewichtung zukünftiger Belohnungen. Umgangssprachlich ausgedrückt beschreib γ die „Sichtweite“ des Agenten. Auch hier gilt $0 < \gamma \leq 1$.

Sowohl α als auch γ werden in anwendungsbezogenen Umgebungen dynamisch an den Lernfortschritt des Agenten angepasst. Die Tatsache, dass in derm nachfolgenden Fallbeispiel beide Hyperparameter statisch gesetzt werden, ist der Einfachheit des Problems geschuldet. Außerdem fällt damit eine Analyse des Einflusses dieser Paramter leichter.

3 Aufgaben

3.1 Mittleres Verschiebungsquadrat

Damit die Diffusionskonstante D in einem diskreten Problem variiert werden kann, muss eine Wahrscheinlichkeit P_{Diff} abhängig von D verwendet werden. Sie gibt an, ob zu einem beliebigen diskreten Zeitpunkt ein Diffusionsschritt stattfindet. Die Diffusion muss im zeitlichen Mittel der Diffusionsgleichung genügen:

$$D = \frac{a^2}{2\bar{\tau}}$$

Dabei ist $a = 1$ die diskrete Schrittweite und $\bar{\tau}$ die mittlere Zeitspanne zwischen zwei Schritten, sodass die Diffusionskonstante zu jedem Zeitpunkt $D = 1/2$ beträgt. Daraus folgt dann

$$P_{\text{Diff}} = \frac{1}{\bar{\tau}} = 2\,D$$

Um die Diffusion auf das Problem zu übertragen, wird eine Funktion erstellt, welche den Zustand mit einer Wahrscheinlichkeit von $P_{\text{Diff}} =$ `P_diffstep` um einen Schritt nach links oder nach rechts verändert.

Die Funktion wird über die Instanz `learner` im main-Skript aufgerufen.

Das zweite Moment der Gauß-Verteilung (also die Varianz σ^2) für den random-walk kann direkt als $2Dt$ abgelesen werden. Da diese Größe linear ist, kann die tatsächlich im Problem vorhandene Diffusionskonstante über einen linearen Fit ermittelt werden. Die Steigung dieses Fits ist das erste Element des zugehörigen Tupels `p = np.polyfit(X,Y,1)`, sodass sich die Diffusionskonstante bestimmen lässt über

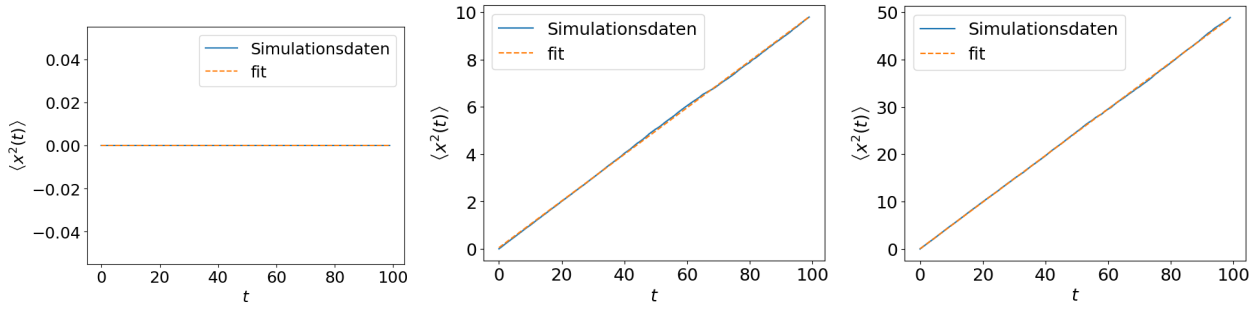
$$D = \texttt{p[0]}/2$$

Um diese Funktionsweise zu testen, werden verschiedene Diffusionskonstanten $D = D_0$ in der Initialisierung der Klasse `agent` definiert. Über die obige Formel werden die Diffusionskonstanten D_1 nach dem linearen Fit gemessen:

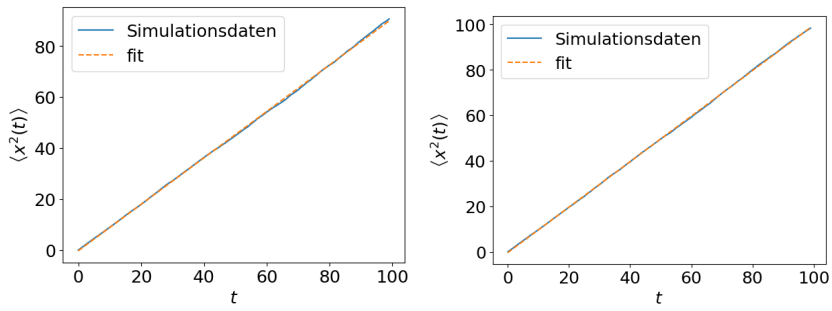
D_0	0	0.05	0.25	0.45	0.5
D_1	0.0	≈ 0.04922	≈ 0.2461	≈ 0.4551	≈ 0.4998

In der vorliegenden Implementation gibt es jedoch Grenzen für mögliche Diffusionskonstanten. Der maximal einstellbare Wert liegt bei $D = 1/2$, sodass $P_{\text{Diff}} = 1$ ist. Für größere D ist die Wahrscheinlichkeit eines Diffusionsschrittes größer als Eins, was physikalisch nicht sinnvoll ist. Der minimal einstellbare Wert liegt bei $D = 0$, womit auch $P_{\text{Diff}} = 0$ folgt. Eine Wahrscheinlichkeit unter Null ist ebenso wenig sinnvoll.

Die Wahl von D wird durch das logische Intervall der Wahrscheinlichkeit $P_{\text{Diff}} \in [0, 1]$ begrenzt.



(a) $D = 0$ **(b)** $D = 0.05$ **(c)** $D = 0.25$



(d) $D = 0.45$ **(e)** $D = 0.5$

Abbildung 1: Linare Fits verschiedener Diffusionskonstanten.

3.2 Implementation des Lernalgorithmus

Im vorliegenden Beispiel sind die Startposition des Agenten sowie die Zielposition fest. Der Agent bewege sich auf einem eindimensionalen „Ring“, sodass periodische Randbedingungen gewährleistet sind. Er kann sich um jeweils eine Position vor- und zurückbewegen oder stehen bleiben. Eine Episode endet dann, wenn der Agent einmal auf dem Ziel verblieben ist.

Damit in jedem vierten Schritt ein zufälliger Diffusionsschritt zu erwarten ist, muss gelten: $P_{\text{Diff}} = 0.25$. Daher wird die Diffusionskonstante auf $D = 0.125$ gesetzt. Um den Lernalgorithmus zu realisieren, sind zunächst einige Funktionen notwendig:

(a) `adjust_epsilon`

Damit der Wert von ε nach 90% der Episoden linear auf Null abgefallen ist, wird ein linearer Ansatz mit einer zusätzlichen Abfrage verwendet.

(b) `choose_action`

Mit einer Wahrscheinlichkeit von ε wird die nächste Aktion zufällig gewählt. Dies geschieht auch dann, wenn die Entscheidungsmatrix mehr als ein Maximum aufweist.

(c) `perform_action`

Um die periodischen Randbedingungen zu wahren, ist an zwei Stellen eine Abfrage des aktuellen Zustandes und gegebenenfalls ein Setzen in den jeweils anknüpfenden Zustand notwendig. Einerseits kann der Zustand durch einen Diffusionsschritt verlassen worden sein und andererseits darf auch keine ausgewählte Aktion das Intervall der Zustände verlassen.

(d) `update_Q`

Um die Einträge der Entscheidungsmatrix bezüglich der eben ausgeführten Aktion im ursprüng-

lichen Zustand zu aktualisieren, wird die Formel des Q-Learning verwendet: Die Belohnung $\mathcal{R}$ ist nur dann nicht verschwindend, wenn der Agent sich auf dem Ziel befindet und dort verweilt.

Da die Anpassung von ε lediglich von der aktuellen Episode abhängt, wird diese vor dem `while`-loop ausgeführt. Die anderen Funktionen müssen in jedem Schritt jeder Episode ausgeführt werden. Anhand des erstellten GIF lässt sich erkennen, dass sich der Agent in der letzten Episode auf direktem Weg zum Ziel bewegt.

Die Einträge der Entscheidungsmatrix $\mathcal{Q}$ lassen neben dem Verlauf des GIF darauf schließen, dass der Lernalgorithmus erfolgreich war:

```
6   [4.52447487e+01 5.74038264e+01 7.13664457e+01]
7   [6.12103562e+01 7.04724078e+01 8.09390808e+01]
8   [7.24101890e+01 8.09870518e+01 8.99936946e+01]
9   [8.09865611e+01 9.99954827e+01 8.09668299e+01]
10  [8.99955048e+01 8.09909496e+01 7.28140006e+01]
11  [8.09955086e+01 7.28055014e+01 6.55018852e+01]
12  [7.28955208e+01 6.54930789e+01 5.89415174e+01]
```

```
66  [5.51370726e-25 3.08560099e-26 1.04955129e-27]
67  [2.35057296e-26 9.93082321e-28 3.96960589e-29]
68  [1.13610653e-27 4.76257653e-29 1.60867827e-29]
69  [1.75348164e-29 1.16195589e-29 2.09484686e-28]
70  [1.03103281e-29 2.19906169e-28 2.36213838e-27]
71  [2.19374839e-28 2.30914011e-27 1.70436437e-26]
72  [1.84718042e-27 1.12766178e-26 9.35098134e-26]
```

Abbildung 2: Einträge der Entscheidungsmatrix $\mathcal{Q}$ im Bereich des Ziels und im Bereich des Minimums.

Zu erkennen sind folgende Zusammenhänge:

- Die `target_position` ist auf 8 gesetzt. Da die Indizierung der States bei Null beginnt, ist der Maximalwert von $\mathcal{Q}$ in der 9. Zeile zu finden. Dieser ist nah am maximal möglichen Wert von 100.
- Links vom Maximum (also in der Zeile darüber) hat die Aktion „rechts“ den höchsten Wert, rechts davon die Aktion „links“.
- Das Minimum von $\mathcal{Q}$ befindet sich in Zeile 69, also in einer Entfernung von $\texttt{N\_states}/2 + 10$ Zuständen. Dieser Wert ist ungenau, da der minimale Wert in etwa bei der Entfernung `N_states` zu erwarten ist.

Insgesamt lässt sich sagen, dass der Lernprozess trotz der Abweichung bei der Entfernung des Minimums als erfolgreich zu bewerten ist. Der Agent hat eine eigenständige Strategie zur Erreichung des Ziels entwickelt, ohne direkt über die Information des Zielzustandes zu verfügen.

3.3 Wahl der Hyperparameter

Die Wahl der Hyperparameter α und γ beeinflusst direkt die mögliche Konvergenz des Algorithmus gegen eine sinnvolle Strategie. Daher werden nachfolgend zwei beispielhafte Untersuchungen bezüglich der Funktion dieser beiden Parameter vorgenommen.

3.3.1 Discount factor γ

Betrachtet wird ein eindimensionales Pfad-Finde-Problem. Die möglichen vier Zustände werden durch die Indizes $0, 1, 2, 3$ charakterisiert und die erlaubten Aktionen sind ein Schritt nach links ($j = 0$) oder ein Schritt nach rechts ($j = 1$). Damit hat die *Learning-Matrix* $\mathcal{Q}$ die Form einer 4×2–Matrix, deren Einträge eindeutig durch die Indizes i und j bezeichnet werden.

Der Agent hat den Startzustand 0 und das Ziel befindet sich im Zustand 3. Bewegt der Agent sich auf das Ziel endet die Epoche und die Belohnung $\mathcal{R}$ wird „ausgeschüttet“. Falls der Agent im Zustand 0 die Aktion 0 wählt, bleibt die Position des Agenten unverändert. Zu Beginn des Lernalgorithmus ist $\varepsilon \approx 1$, sodass die Aktionen zufällig gewählt werden.

Da zu Beginn alle Einträge Null sind ($Q_{ij} = 0$), verändert nach Gleichung (1) nur ein Schritt auf das Ziel einen einzigen Eintrag in Q:

$$Q_{21}^{\text{neu}} = Q_{21}^{\text{alt}} + \alpha\left(\mathcal{R} + \gamma \max_k(Q_{3k}) - Q_{21}^{\text{alt}}\right) = 0 + \alpha(\mathcal{R} + \gamma \cdot 0 - 0) = \alpha\mathcal{R}$$

Damit hat die Matrix $\mathcal{Q}$ nach der ersten Episode folgende Gestalt:

$$\mathcal{Q} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & \alpha\mathcal{R} \\ 0 & 0 \end{pmatrix}$$

Nun wird die Schrittfolge $\rightarrow\leftarrow\rightarrow\rightarrow\rightarrow$ betrachtet. Analog zur vorherigen Begründung liefert der erste Schritt $\mathcal{Q}_{01}^{\text{neu}} = 0$. Ein Schritt „zurück“ verändert die $\mathcal{Q}$-Matrix ebenfalls nicht, also $\mathcal{Q}_{10}^{\text{neu}} = 0$. Ein erneuter Schritt nach rechts lässt $\mathcal{Q}$ wieder unverändert, erst der Schritt auf das Feld 2 verschwindet nicht:

$$\mathcal{Q}_{11}^{\text{neu}} = \mathcal{Q}_{11}^{\text{alt}} + \alpha\left(\gamma \max_k(\mathcal{Q}_{2k} - \mathcal{Q}_{11}^{\text{alt}})\right) = \alpha^2\gamma\mathcal{R}$$

Schließlich folgt der letzte Schritt auf das Ziel, sodass

$$\mathcal{Q}_{21}^{\text{neu}} = \mathcal{Q}_{21}^{\text{alt}} + \alpha\left(\mathcal{R} + \gamma \max_k(\mathcal{Q}_{3k}) - \mathcal{Q}_{21}^{\text{alt}}\right) = \alpha\mathcal{R} + \alpha(R - \alpha\mathcal{R}) = 2\alpha\mathcal{R} - \alpha^2\mathcal{R}$$

und die Matrix $\mathcal{Q}$ hat die Form

$$\mathcal{Q} = \begin{pmatrix} 0 & 0 \\ 0 & \alpha^2\gamma\mathcal{R} \\ 0 & 2\alpha\mathcal{R} - \alpha^2\mathcal{R} \\ 0 & 0 \end{pmatrix}$$

Beginnt nun die dritte Eipsode mit dem Schritt $\rightarrow$, so ist

$$\mathcal{Q}_{01}^{\text{neu}} = \mathcal{Q}_{01}^{\text{alt}} + \alpha\left(\gamma \max_k(\mathcal{Q}_{1k} - \mathcal{Q}_{01}^{\text{alt}})\right) = \alpha^2\gamma^2\mathcal{R}$$

und $\mathcal{Q}$ ist damit:

$$\mathcal{Q} = \begin{pmatrix} 0 & \alpha^2\gamma^2\mathcal{R} \\ 0 & \alpha^2\gamma\mathcal{R} \\ 0 & 2\alpha\mathcal{R} - \alpha^2\mathcal{R} \\ 0 & 0 \end{pmatrix}$$

Man sieht: der „Erfolg“ eines Schrittes überträgt sich auf die Entscheidung im vorausgehenden Zustand. Schon zu Beginn der dritten Episode liegt somit im Ausgangszustand eine Information zur „richtigen“ Entscheidungsfindung vor. Der Agent lernt also, bei $i = 0$ nach rechts zu gehen.

In dem hier betrachteten Modell kann der Agent jedoch nicht lernen, nach links zu laufen. Denn aufgrund der fehlenden Periodizität kann er das Zielfeld, auf dem der *target reward* $\mathcal{R}$ ausgeschüttet wird, nie mit einem Schritt nach links erreichen. Damit bleiben stets alle Matrixelemente der ersten Spalte (also der Aktion $j = 0$) gleich Null.

3.3.2 Learning rate α

Die *learning rate* bestimmt, wie schnell der Algorithmus konvergiert. Um die verschiedenen Größen des Parameters α genauer zu betrachten, werden einige Fälle untersucht. Dazu wird die Diffusionskonstante in diesem Abschnitt auf Null gesetzt.

Um die Performance des Algorithmus im Verhältnis zur Simulationszeit abzubilden, wird eine Lernkurve angelegt. Diese beinhaltet für jede Epsiode die Anzahl der Schritte, bis das Ziel erreicht wurde.

Zunächst wird die *learning curve* für die ursprüngliche *learning rate* $\alpha = 0.01$ dargestellt, bei der der Agent jede Episode auf einer festen Startposition beginnt:

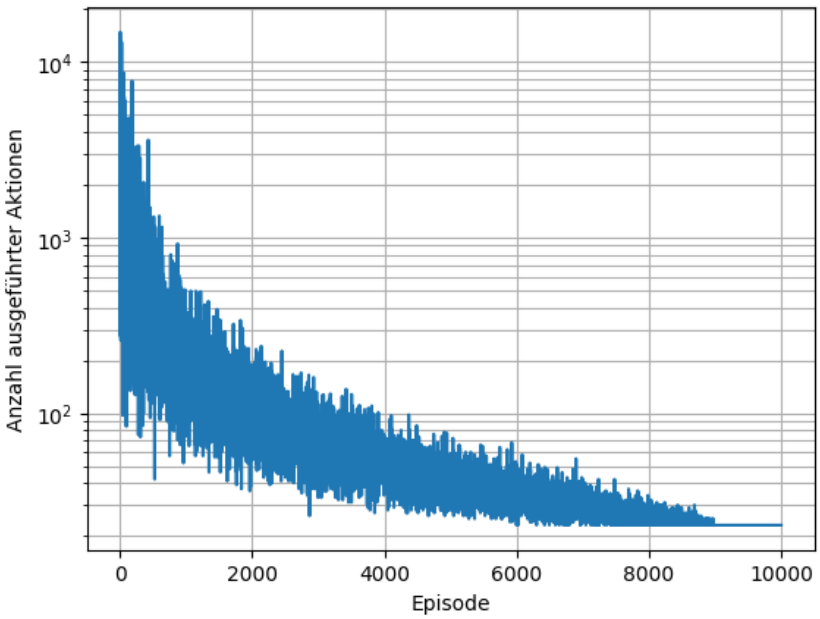


Abbildung 3: Anzahl der benötigten Schritte bei einer *learning rate* von $\alpha = 0.01$.

Es ist zu erkennen, dass die Anzahl benötigter Schritte exponentiell abfällt. Sobald $\varepsilon = 0$ ist, liegt ein konstanter Wert vor. Dieser Wert entspricht gerade der minimalen Entfernung von Start- und

Zielposition.

Nun wird die Startposition des Agenten zu Beginn jeder Episode zufällig festgelegt. Die *learning curve* verändert sich folgendermaßen:

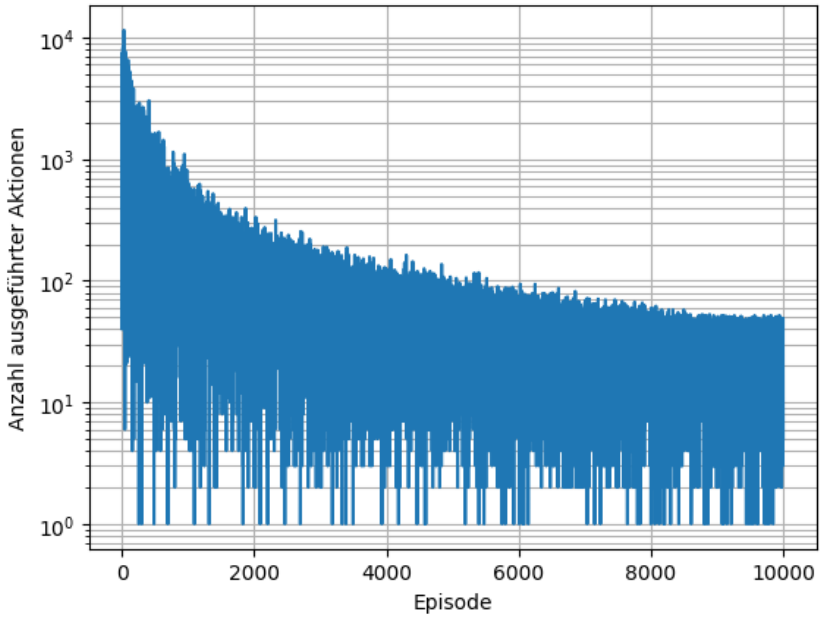


Abbildung 4: Anzahl der benötigten Schritte bei einer *learning rate* von $\alpha = 0.01$ und zufälliger Startposition.

Hier konvergiet die *learning curve* nicht gegen einen konstante Wert, da trotz $\varepsilon = 0$ (also Konvergenz) stets eine zufällige Startposition vorliegt.
Damit das Verhalten des Agenten dennoch beurteilt werden kann, wird anstatt der Anzahl der benötigten Schritte das Verhältnis der benötigten Aktionen zur minimal möglichen Aktionszahl dargestellt:

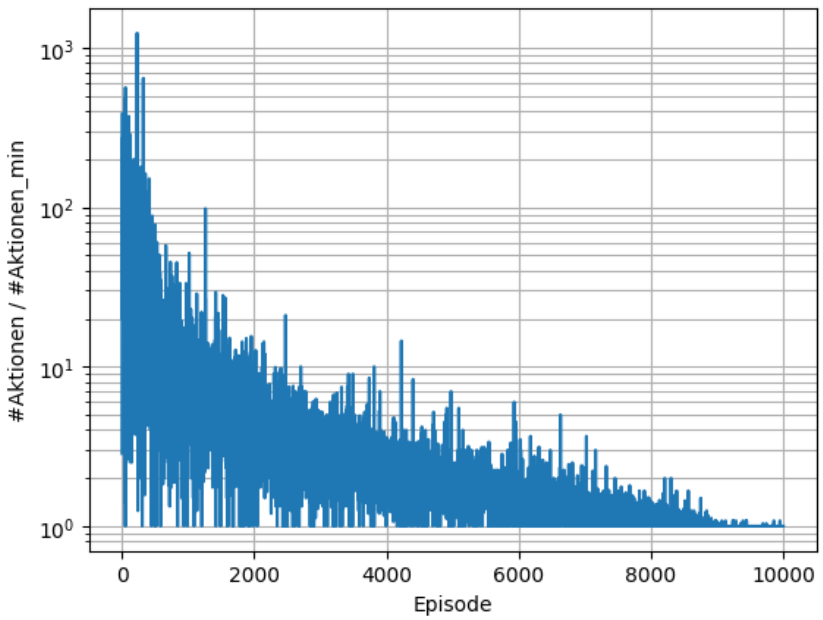


Abbildung 5: Verhältnis von durchgeführten zu minimal benötigen Aktionen (bzw. Schritten) bei einer *learning rate* von $\alpha = 0.01$ und zufälliger Startposition.

Hier ist zu beachten, dass sich die Anzahl der benötigten Aktionen und der benötigten Schritte um genau Eins unterscheidet, da der Agent zum Beenden der Episode einmal im Zielzustand verweilen muss.

Es ist zu sehen, dass die *learning curve* nun gegen Eins konvergiert. Um nun die Auswirkungen der *learning rate* genauer zu untersuchen, wird $\alpha = 0.999999$ gesetzt. Dies ist der maximale Wert, denn im Formalismus des $\mathcal{Q}$-Learning gilt $0 < \alpha < 1$. Der Plot hat für die (nach wie vor hohe) Anzahl von 10^4 Epsioden folgende Gestalt:

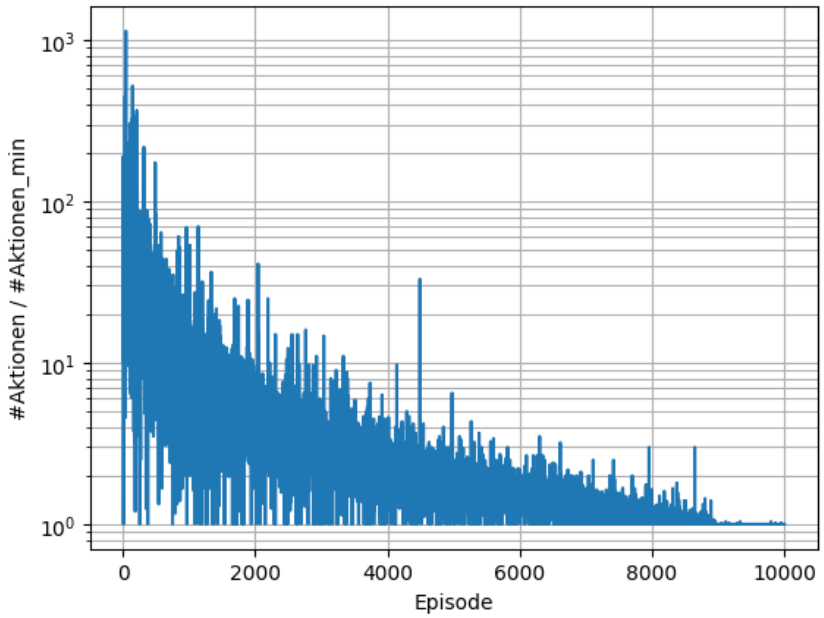


Abbildung 6: Verhältnis von durchgeführten zu minimal benötigen Aktionen (bzw. Schritten) bei einer *learning rate* von $\alpha = 0.999999$ und zufälliger Startposition.

Entgegen der Erwartung ist kein Unterschied zum vorherigen Fall mit $\alpha = 0.01$ zu erkennen. Dies kann entweder der Einfachheit des betrachteten Problems oder aber auch an der hohen Anzahl von Epsisoden geschuldet sein. Dieses Verhalten lässt sich auch für eine geringe Anzahl von Episoden beobachten.
Dennoch ist zu erwarten, dass die Wahl von α für kompliziertete Probleme ausschlaggebend ist.

3.4 Stochastisches Hindernis

In diesem Abschnitt werden für den Agenten Hindernisse hinzugefügt, welche die Aktion des Agenten entgegen dieser Hindernisse zurücksetzen können. Betrachtet werden 15 mögliche eindimensionale Zustände, wobei der Agent im Zustand 8 startet und das Ziel im Zustand 12 erreichen soll. Im Intervall $[9, 12]$ können die stochastischen Hindernisse auftreten. Diese setzen den Agenten mit einer vorgegebenen Wahrscheinlichkeit um einen Zustand zurück, sobald sie erreicht werden.

Im modifizierten Quellcode wird die gesamte Verschiebung des Agenten durch Aktionen festgehalten, sobald dieser trainiert ist. Über das Vorzeichen dieser Verschiebung durch Aktionen lässt sich erkennen, welche Strategie der Agent gewählt hat.
Es ist zu untersuchen, ab welcher Hinderniswahrscheinlichkeit der Agent die Strategie ändert und den *langen* Weg wählt, in welchem die Hindernisse umgangen werden können.

Damit der Agent die Strategie ändert, muss die Anzahl an nötigen Aktionen durch die Hindernisse über den *kurzen* Weg N_k größer werden als die Anzahl der Aktionen N_l über den *langen* äußeren Weg:

- *langer* Weg: $N_l = 8 + (15 - 12) + 1 = 12$
- *kurzer* Weg ohne Hindernisse: $N_k = 12 - 8 = 4$

Die Hindernisse müssen also dafür sorgen, dass der Agent im Mittel $N_l - N_k = 8$ Schritte mehr benötigt. Sobald dies erreicht ist, sollte der Agent die Strategie wechseln. Die Wahrscheinlichkeit P muss also mindestens so hoch sein, dass der Agent 8 mal oder mehr um einen Zustand zurückgesetzt wird:

$$\Delta N = (3p)^3 \quad \text{und} \quad \Delta N \geq 8 \quad \text{sodass} \quad p \geq \frac{\sqrt[3]{8}}{3} = 2/3 \approx 0.66$$

Damit ist eine Wahrscheinlichkeit von grob $P \approx 70\%$ zu erwarten.

Entgegen der Erwartung ließen sich im Versuch andere Werte für P beobachten, es zeigt sich ein Wechsel der Strategie bei einer Hinderniswahrscheinlichkeit von über 99%:

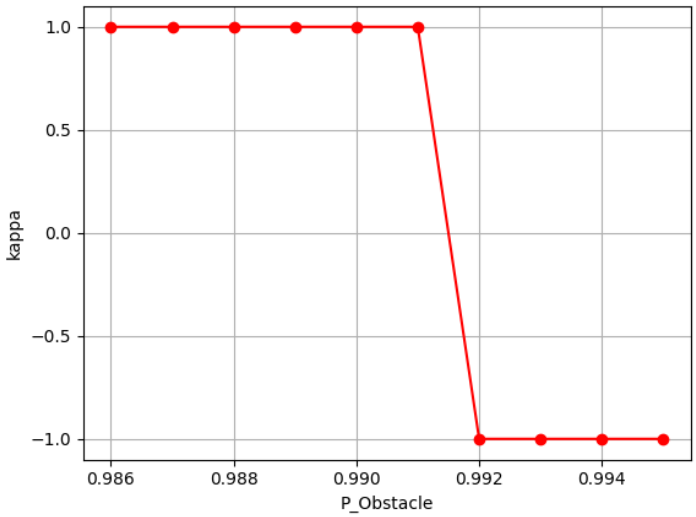


Abbildung 7: Übergang von κ bei $P \approx 99.1\%$.

Diese Beobachtung ist vermutlich auf eine fehlerhafte Implementation im Quellcode zurückzuführen, welche trotz mehrmaligem vollständigen Neuschreiben des Programms nach wie vor auftritt.

Eine separater, eigens erstellter Quellcode zur Analyse der Hinderniswahrscheinlichkeit ist angehangen (siehe `p_analysis.py`). Dieser Algorithmus liefert über eine hohe Zahl von Durchläufen eine mittlere Wahrscheinlichkeit von $P \approx 71\%$, welche sich mit der oben geschilderten Erwartung deckt.

Außerdem ist anzumerken, dass wie auch im vorherigen Abschnitt eine erhebliche Veränderung der *learning rate* α keine Auswirkung auf die Strategie des Agenten hat. Auch hier kann bereits die Einfachheit des Problems der Grund dafür sein.