Karlsruher Institut für Technologie

Grundlagen der Künstlichen Intelligenz

Wintersemester 24/25

Vorlesung 13:

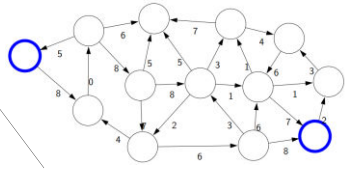
Markov Decision Processes

Prof. Dr. Gerhard Neumann

Vorlesung 1

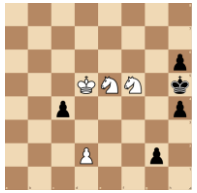
Modellierung

KI



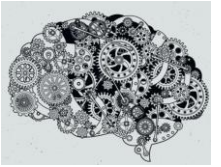
Schlussfolgerung — **Lernen**

Reflex



Zustände

Variablen

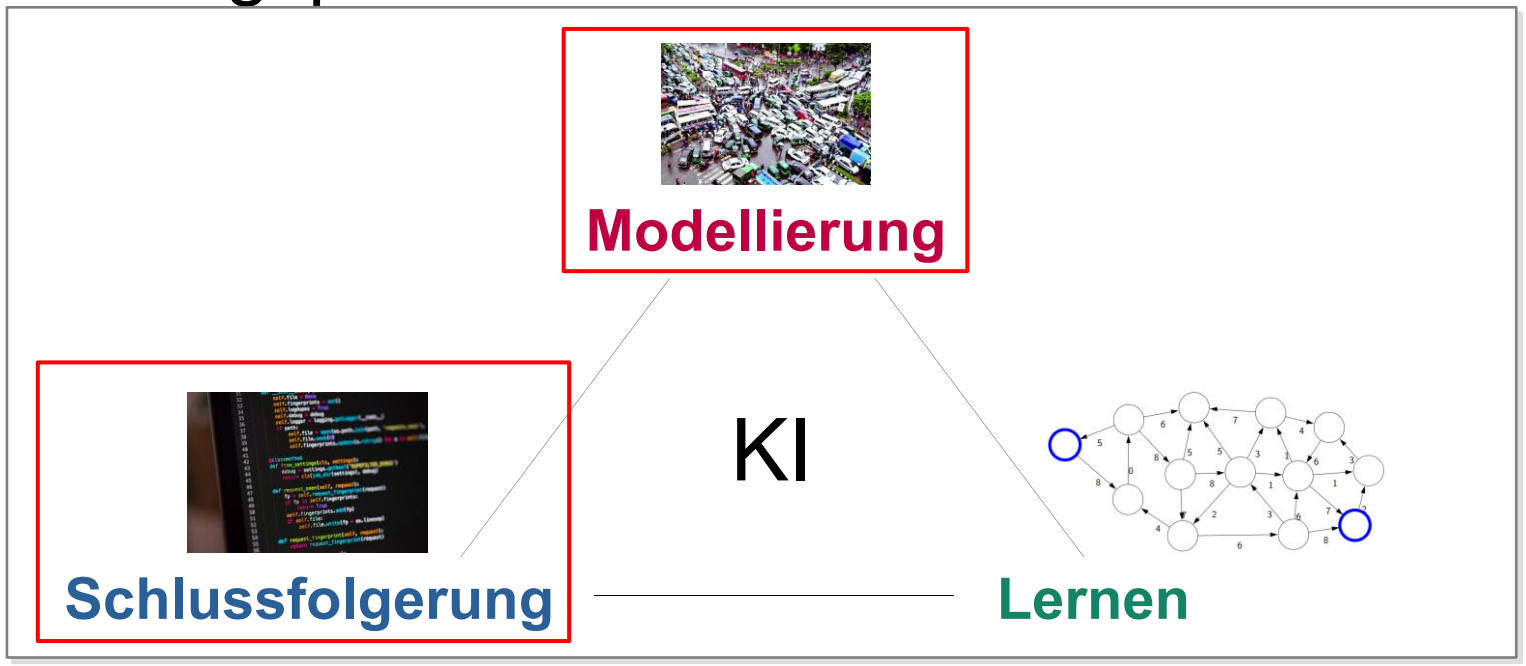


Logik

"niedrige Intelligenz" — "Intelligenz auf hohem Niveau"

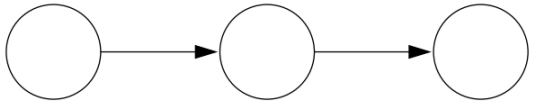
Künstliche Intelligenz

Markov-Entscheidungsprozesse



Zustandsbasierte KI

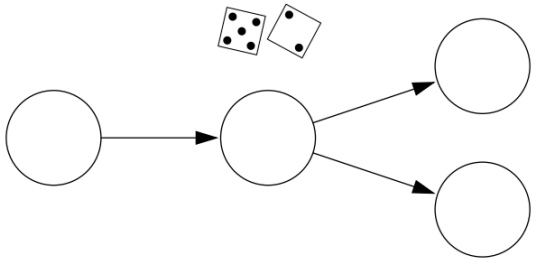
Suchprobleme: Volle Kontrolle



z.B. Routenfindung

Markov-Decision-Prozesse: Gegen die Umgebung

- Zustandsübergänge sind unsicher



z.B. Blackjack

Adversarial Games: Gegen einen Gegner

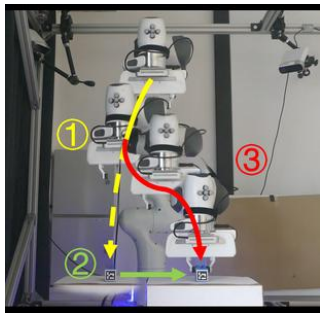
z.B. Schach

Reflex **Zustände** Variablen Logik

"niedrige Intelligenz" "Intelligenz auf hohem Niveau"

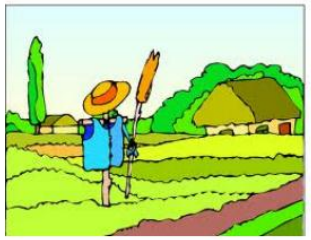
Künstliche Intelligenz

Anwendungen



- **Robotik:** Entscheidung, wohin man sich bewegen soll, Ungewissheit in der Umgebung (Aktoren können ausfallen, auf Hindernisse treffen, ...)

- **Ressourcenzuweisung:** Entscheidung, was produziert werden soll, ohne Kenntnis der Kundennachfrage nach verschiedenen Produkten



- **Landwirtschaft:** entscheiden, was gepflanzt werden soll, kennen aber das Wetter und damit den Ernteertrag nicht

- **Spiele:** Entscheiden Sie, wie Sie spielen wollen, kennen Sie die stochastischen Ereignisse und das Verhalten des Gegners nicht

Deep Reinforcement Learning für Robotik

Dextile manipulation:

- OpenAI 2017/2018
- Learn a DNN model
- Plan / model predictive control using shooting methods

Laufroboter:

- Lernen in Simulation von Laufbewegungen – Übertragung auf den echten Roboter

Geschichte: Markov-Entscheidungsprozesse

- MDPs: Mathematisches Modell für die Entscheidungsfindung unter Unsicherheit.
- MDPs wurden erstmals in den 1950er und 1960er Jahren eingeführt.
- Der Begriff "Markov" bezieht sich auf Andrey Markov, da MDPs Erweiterungen von Markov-Ketten sind und Entscheidungen (Handlungen oder Wahlmöglichkeiten) ermöglichen.

Agenda für heute...

Suchbäume mit Zufallsknoten:

- Expectimax-Bäume

Markov-Entscheidungsprozesse (MDPs):

- Definition, Discounting und Optimalität
- Value-Funktionen und State-Action-Valuefunktionen

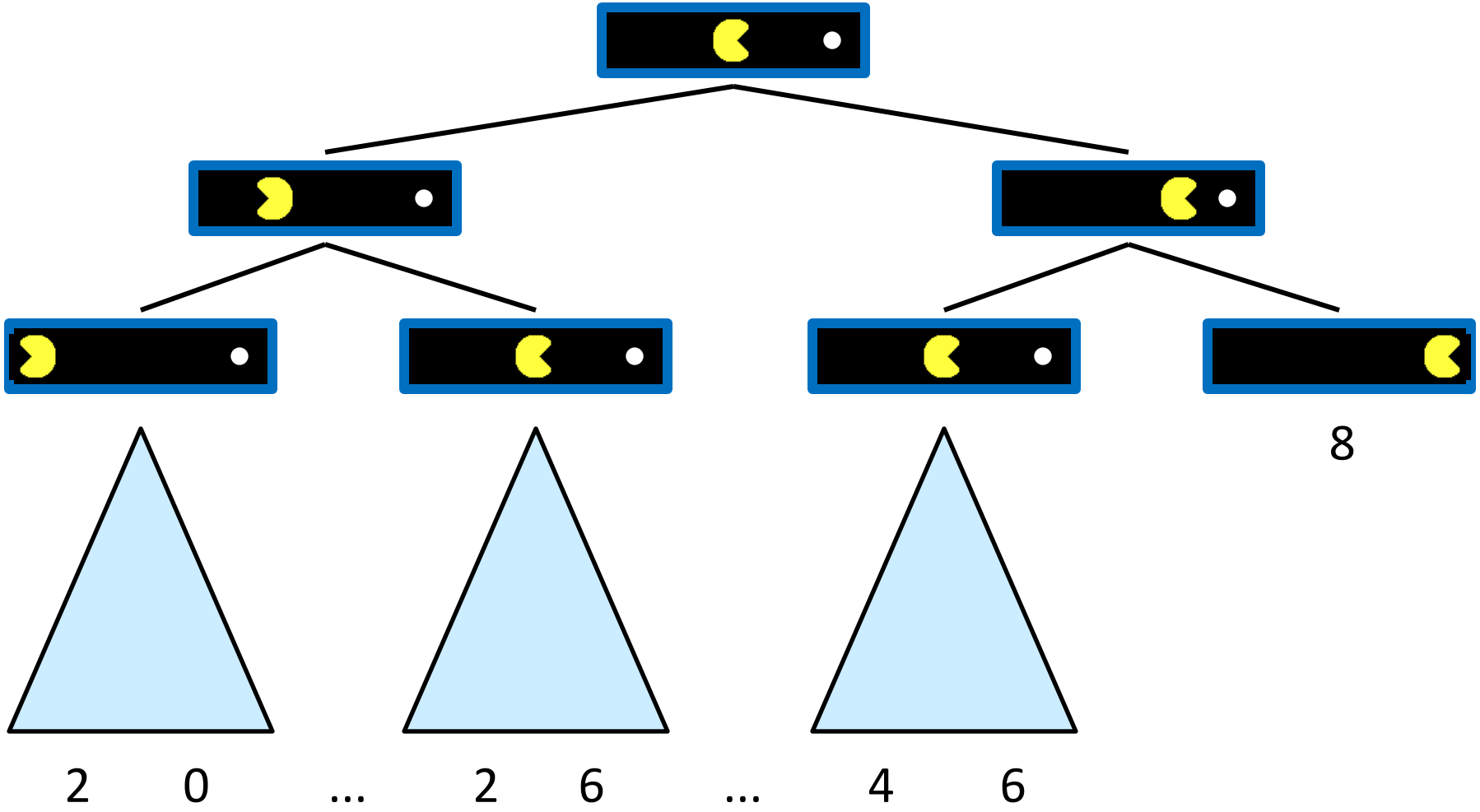
Lösen von MDPs:

- Value Iteration

Bildnachweis: Dan Klein und Pieter Abbeel, UC Berkeley

Suchbäume mit Zufallsknoten

Zurück zu Suchbäume



Values

Wert (“Value”) eines Zustands:

- Das beste erreichbare Ergebnis (Nutzen) in diesem Zustand

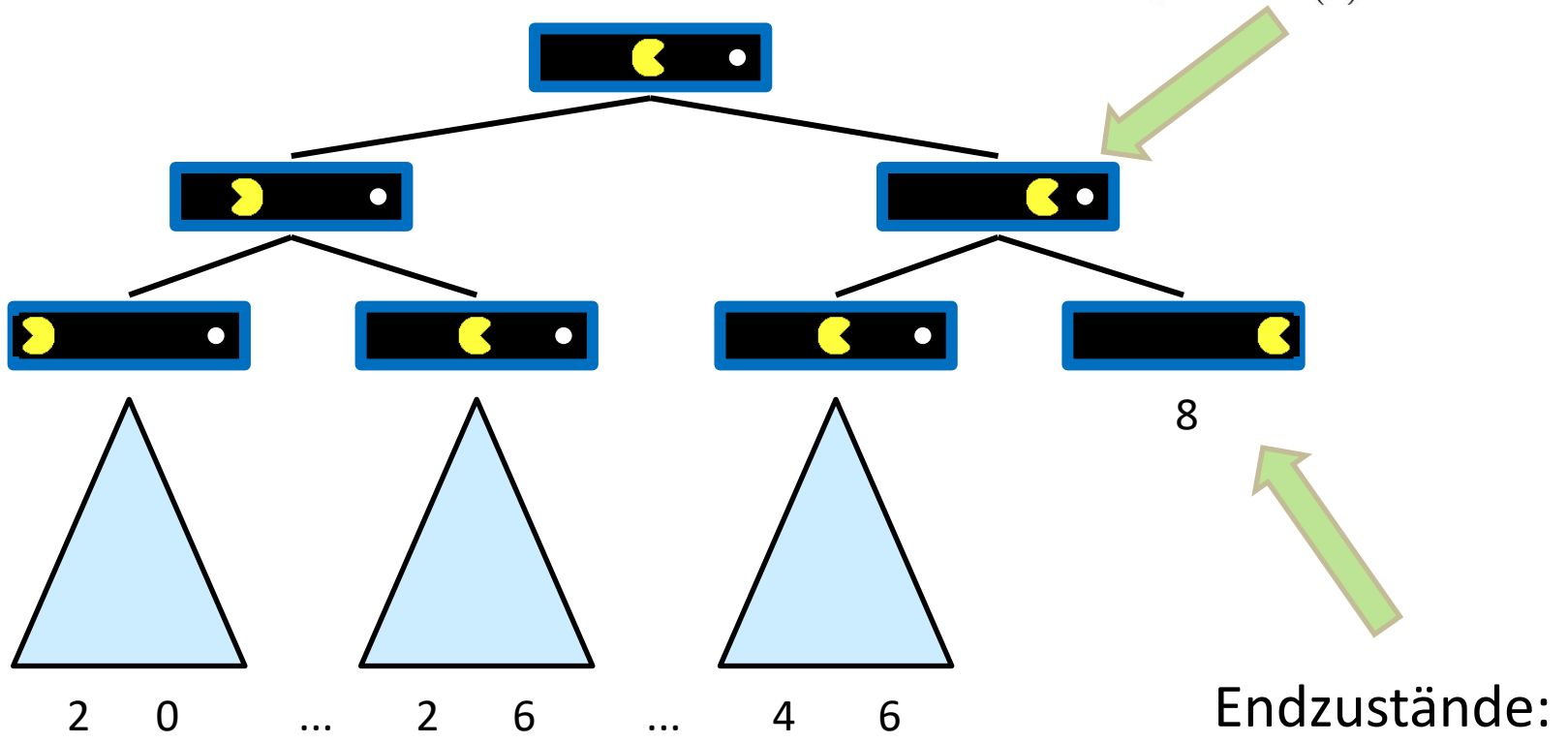
Zwischenknoten:

$$V(s) = \max_{s' \in \text{children}(s)} V(s')$$

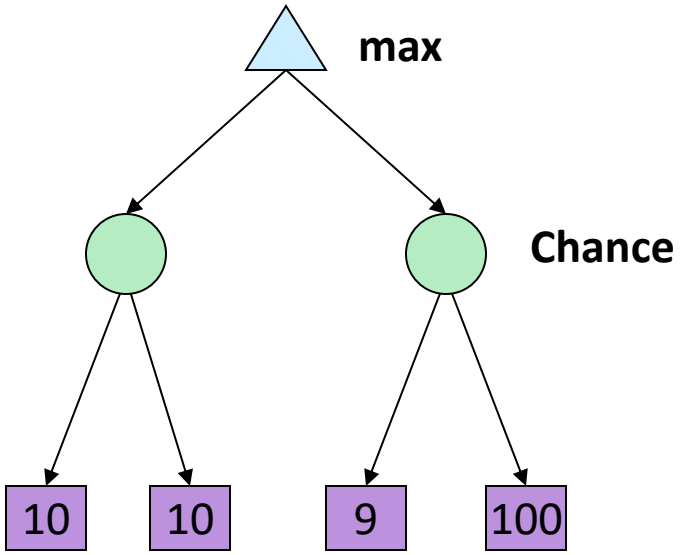
- Der Value ist durch den Höchstwert aller Nachfolgezustände gegeben

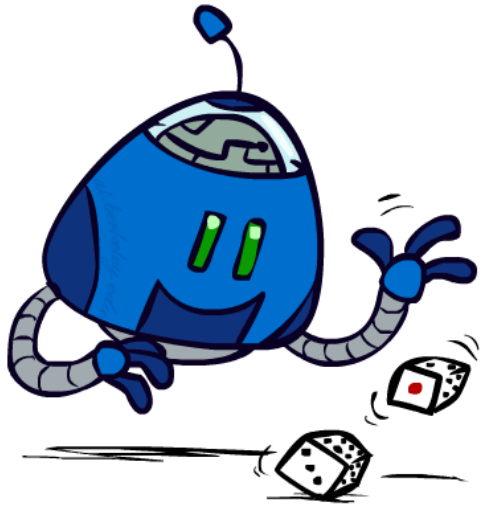
Endzustände:

$$V(s) = \text{known}$$



Spielen gegen eine Umgebung





Ergebnisse sind nicht vollständig kontrollierbar, können aber durch Zufall kontrolliert werden

Expectimax Suche

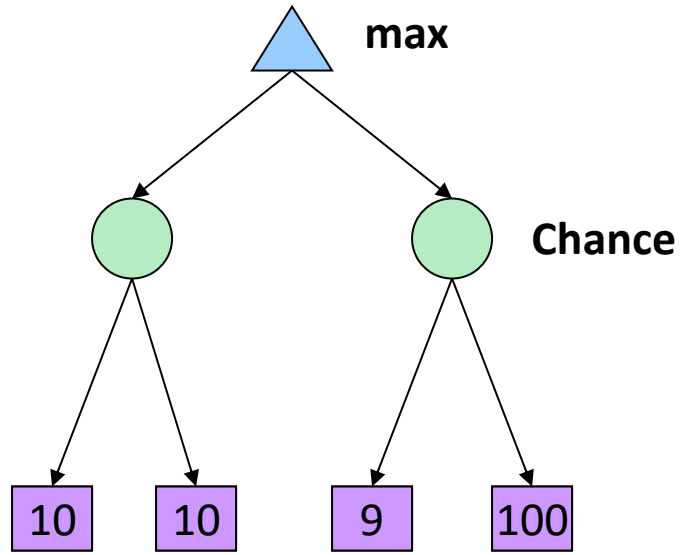
Warum sollten wir nicht wissen, was das Ergebnis einer Handlung sein wird?

- Explizite Zufälligkeit: Würfeln
- Aktionen können fehlschlagen: Beim Bewegen eines Roboters können die Räder durchdrehen.

Die Values sollten nun die Ergebnisse **im erwarteten Fall** widerspiegeln**.**

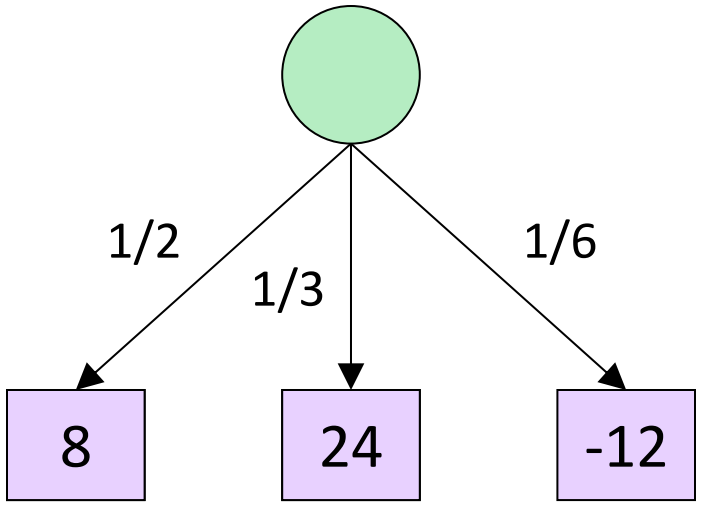
Expectimax-Suche: Berechnung der durchschnittlichen Punktzahl bei optimalem Spiel

- Max Knoten wie bei der Suche
- Die Zufallsknoten berechnen den erwarteten Nutzen
- D.h. gewichteter Durchschnitt (Erwartung) der Kinder



Expectimax Pseudocode

```
def exp-value(state):
    v = 0 initialisieren
    für jeden Nachfolger des Staates:
        p = Wahrscheinlichkeit(Nachfolger)
        v += p * Wert(Nachfolger)
    Rückgabe v
```



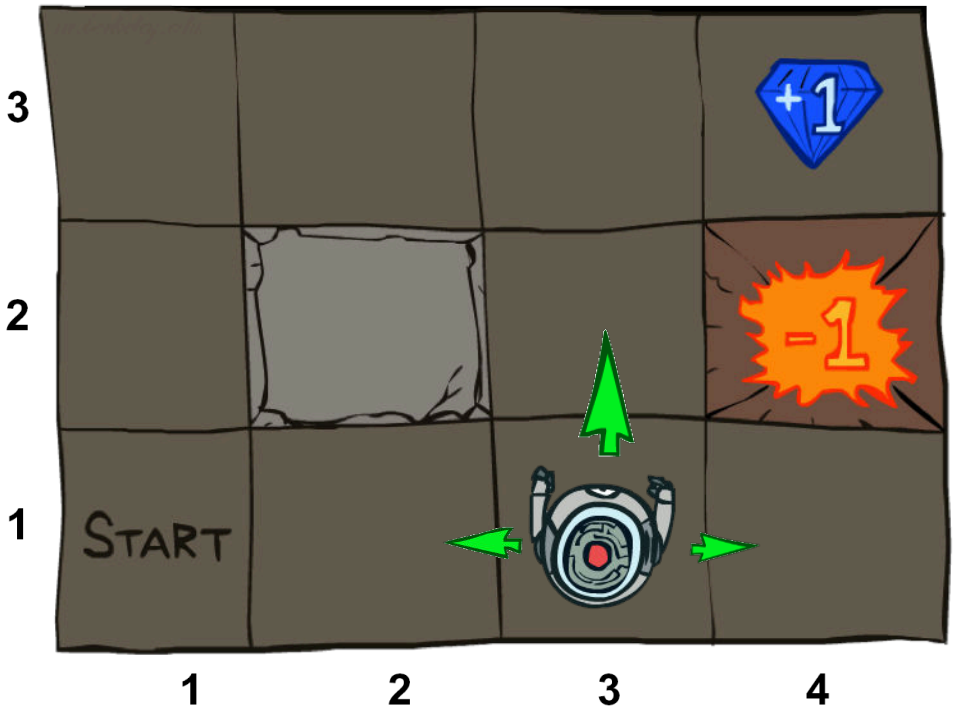
$$v = (1/2)\,(8) + (1/3)\,(24) + (1/6)\,(-12) = 10$$

Markov Decision Processes

Markov-Entscheidungsprozesse

Ein MDP ist definiert durch:

- Eine Menge von Zuständen (States) s $\in$ S
- Eine Menge von Aktionen a $\in$ A
- Eine Transitionsverteilung p(s'| s, a)
  - Wahrscheinlichkeit, dass a von s nach s' führt
  - Wird auch das “Modell” oder die “Dynamik” genannt
- Eine Belohnungsfunktion r(s, a)
- Eine Verteilung $\mu_0(s)$ über den Startzustand
- Eine optionale Liste von Endzuständen



Beispiel: Grid World

Ein labyrinthartiges Problem

- Der Agent lebt in einer Grid-World
- Mauern versperren dem Agenten den Weg

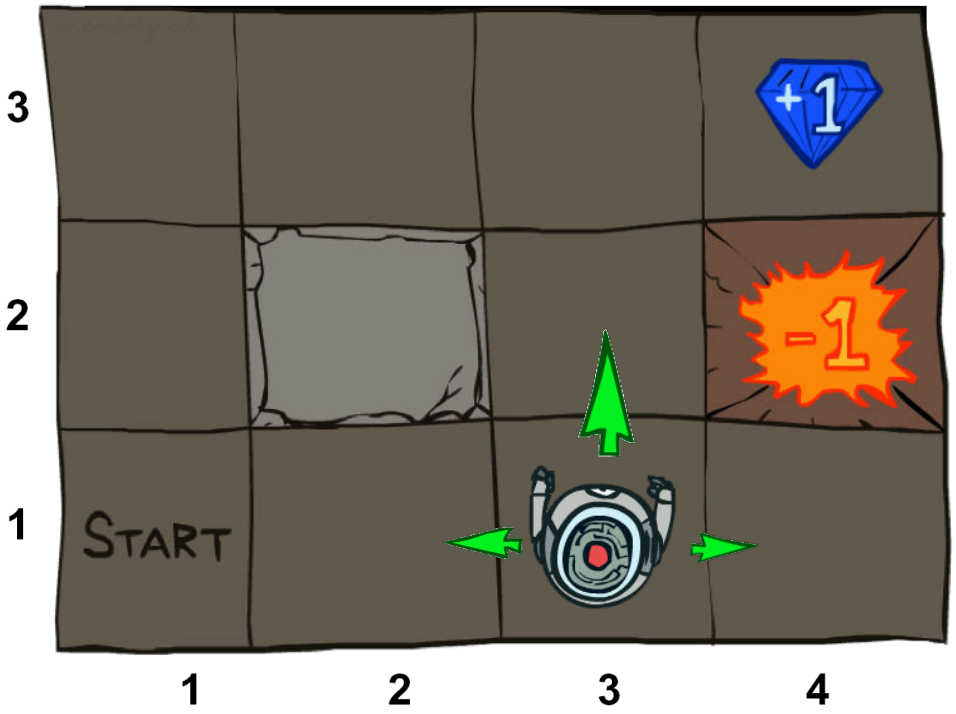
Unsichere Bewegung: Aktionen verlaufen nicht immer wie geplant

- In 80% der Fälle führt die Aktion Nord den Agenten nach Norden (wenn sich dort keine Wand befindet)
- In 10 % der Fälle nimmt Aktion “Nord” den Agenten nach Westen, in 10 % nach Osten.
- Befindet sich eine Wand in der Richtung, in die der Agent gegangen wäre, bleibt der Agent an Ort und Stelle.

Der Agent erhält in jedem Zeitschritt Rewards

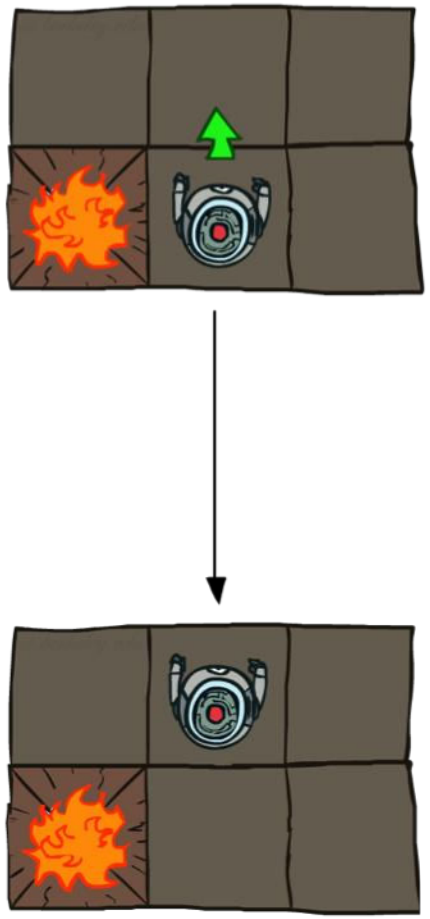
- Kleiner Reward für jeden Schritt (kann negativ sein)
- Großer Reward kommen am Ende (gut oder schlecht)

Ziel: Maximierung der Summe der Rewards

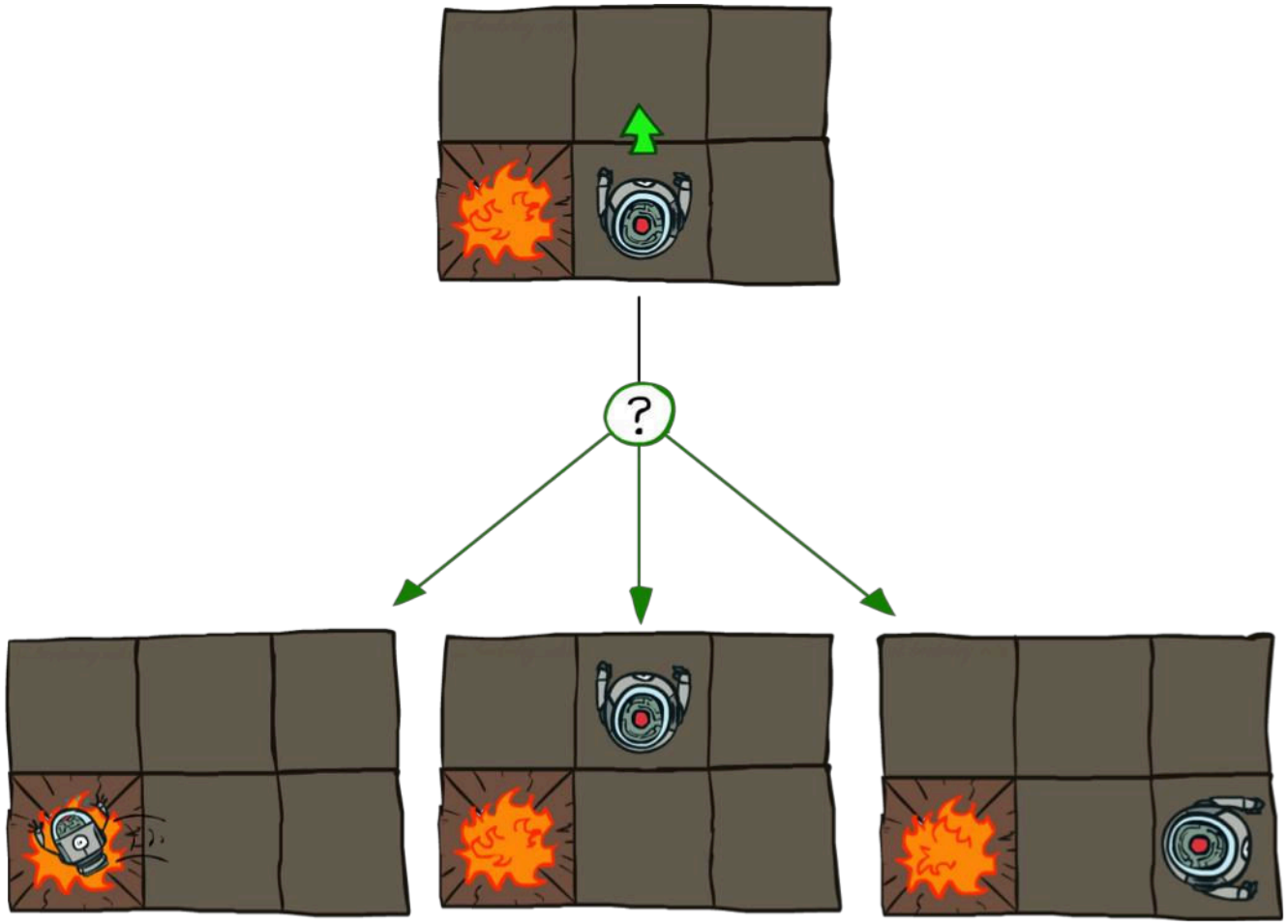


Grid World-Aktionen

Deterministische Welt



Stochastische Welt



Was ist Markov an MDPs?

- "Markov" bedeutet im Allgemeinen, dass gegeben des gegenwärtigen Zustands, die Zukunft von der Vergangenheit unabhängig ist.
- Daher: Die Ergebnisse von Aktionen hängen nur vom aktuellen Zustand ab, nicht von der Vergangenheit

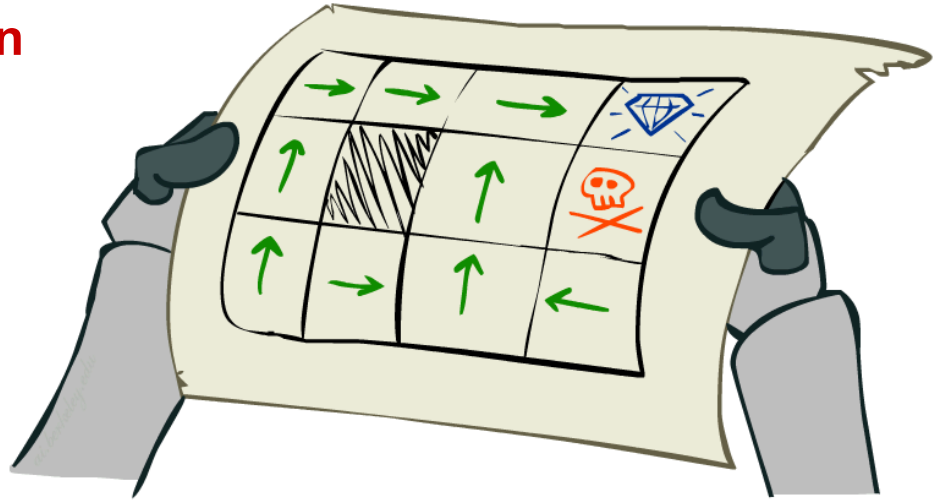
$$p(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \dots S_0 = s_0)$$
$$=$$
$$p(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

- Genau wie bei der Suche, bei der die Nachfolgefunktion nur vom aktuellen Zustand abhängen kann (nicht von der Historie)

Andrej Markow
(1856-1922)

Policies

- **Für MDPs wollen wir eine optimale Policy π^*: S → A finden**
  - Eine Policy π gibt eine Aktion für jeden Zustand vor
  - Eine optimale Policy ist eine Policy, die den erwarteten Reward maximiert, wenn sie befolgt wird.



- **2 Arten von Policies**
  - **Stochastisch:**

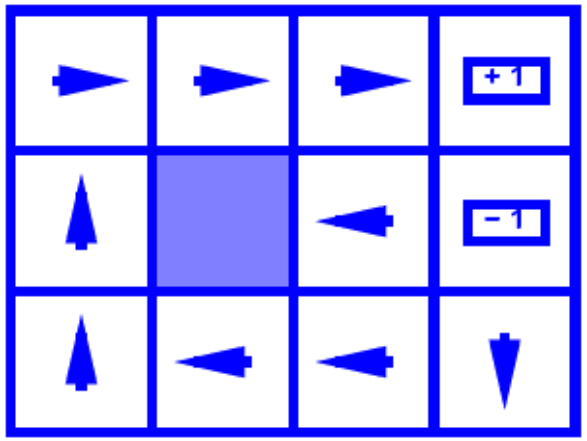
$$\boldsymbol{a}_t \sim \pi(\boldsymbol{a}|\boldsymbol{s})$$

    - Dient der Exploration (wird später zum Lernen benötigt)
  - **Deterministisch:**

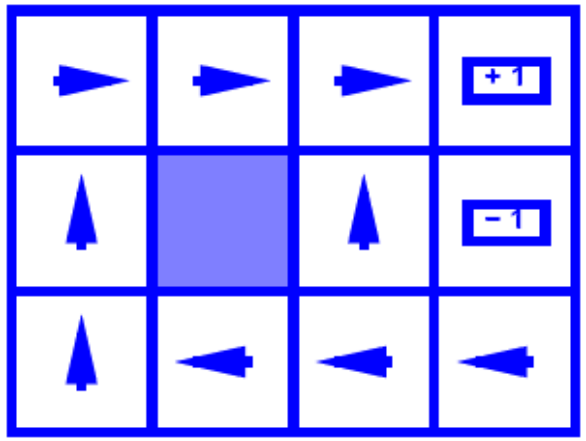
$$\boldsymbol{a} = \pi(\boldsymbol{s})$$

    - Es gibt immer eine deterministische optimale Strategie für einen MDP
    - Sonderfall einer "stochastischen Policy" mit $\pi(\boldsymbol{a}_t|\boldsymbol{s}) = 1$

Beispiele für optimale Policies

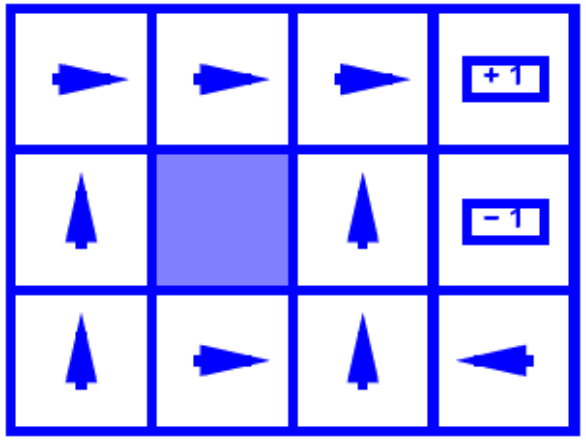


r(s) = -0,01

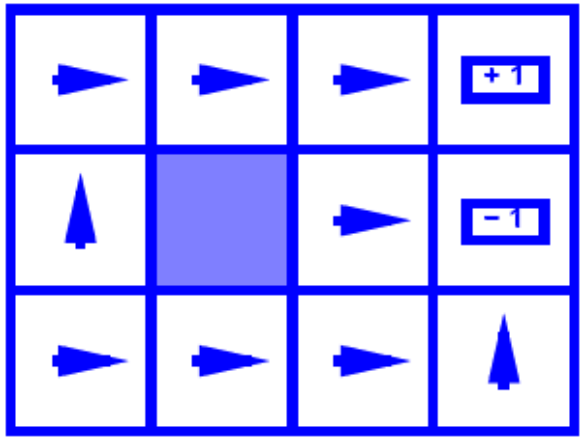


r(s) = -0,03

„Living“ Reward



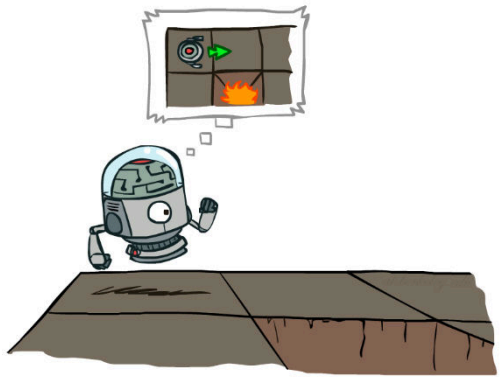
r(s) = -0,4

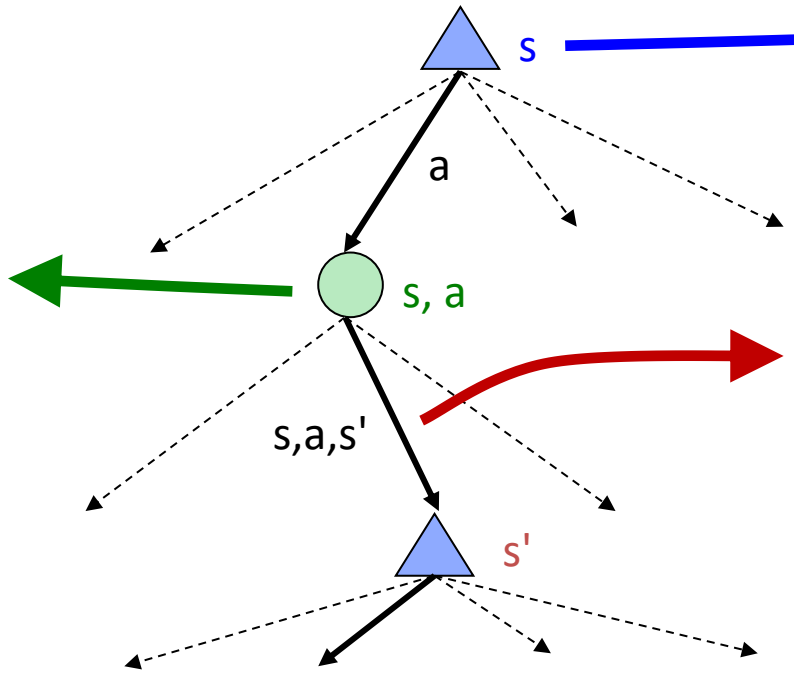


r(s) = -2,0

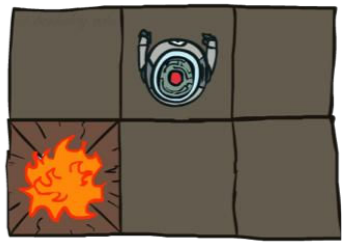
MDP-Suchbäume

Jeder MDP-Zustand projiziert einen expectimax-ähnlichen Suchbaum





s ist ein *Zustand*



(s,a,s') Transition

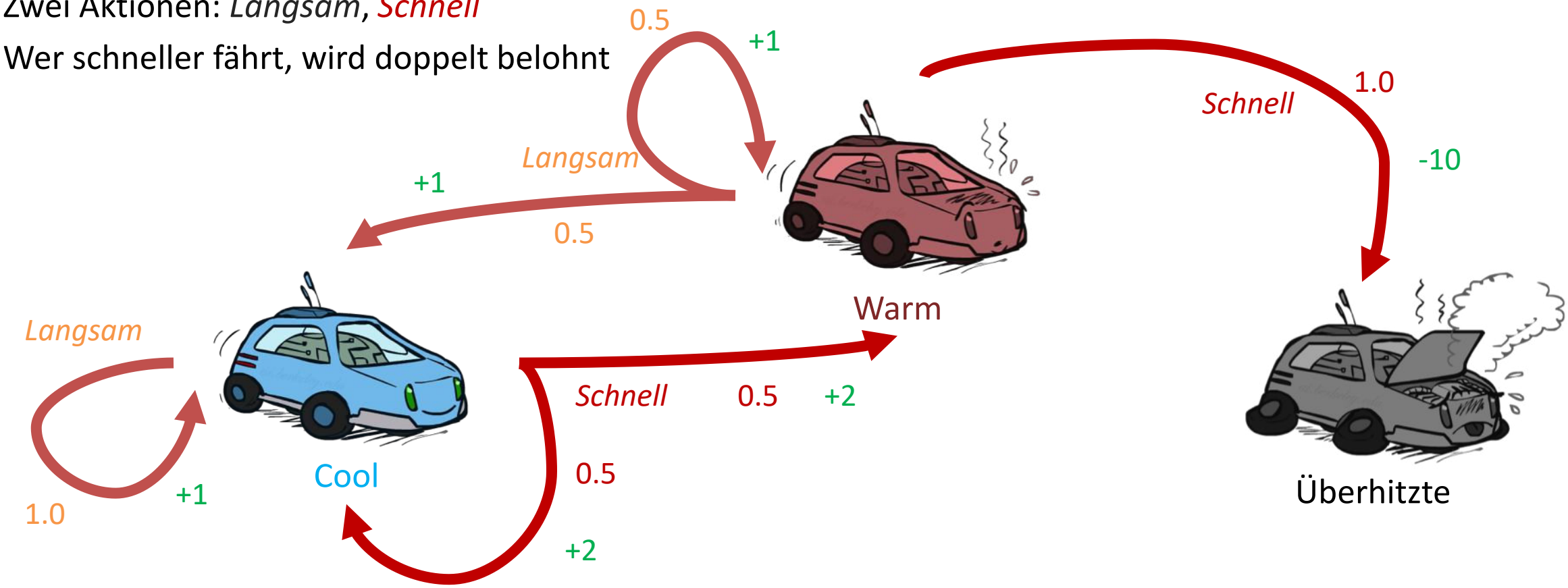
p(s'|s,a)

r(s,a,s')

- **Kleine Änderung:** Berücksichtigen Sie, dass es auch Rewards auf dem Weg gibt
- Es ist jedoch sehr ineffizient, MDPs so zu lösen
- Keine Wiederverwendung von Berechnungen, der Baum muss von jedem Zustand aus berechnet werden

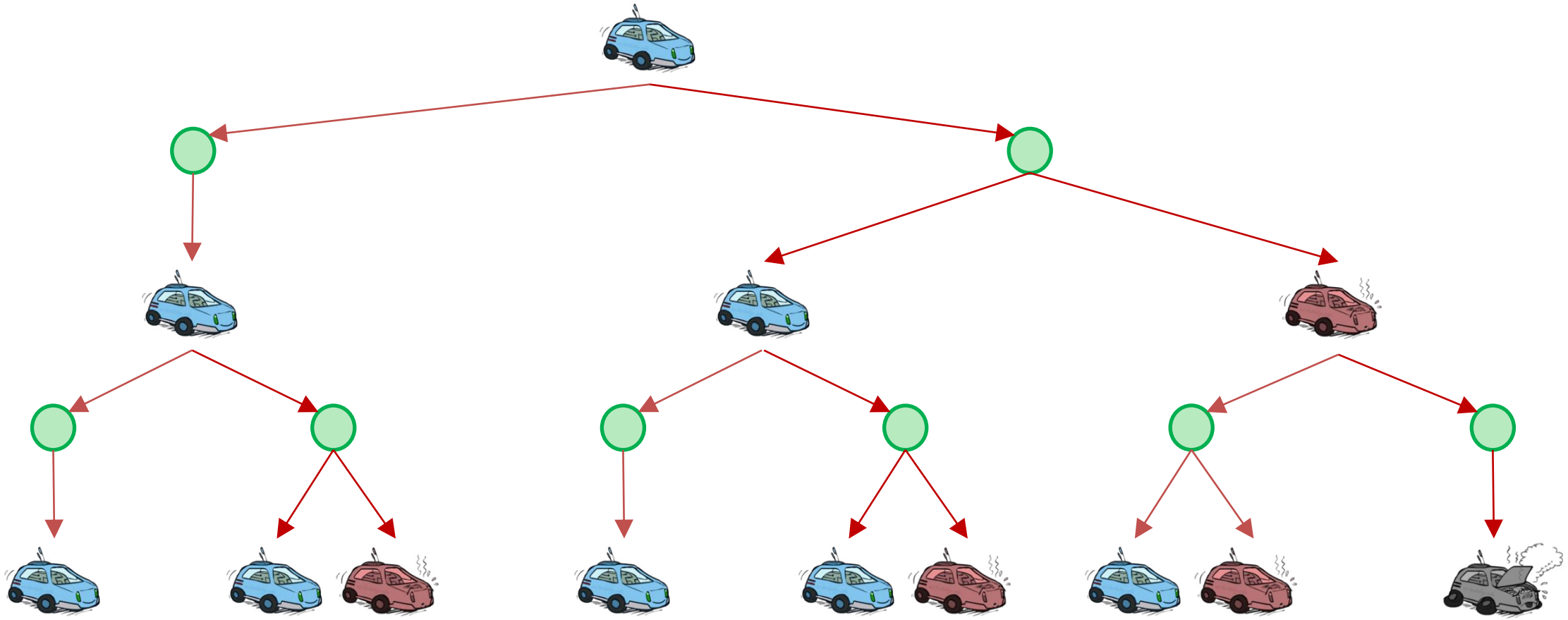
Beispiel: Auto-Rennen

- Ein Roboterauto will schnell fahren, ohne zu überhitzen
- Drei Zustände: Kühl, Warm, Überhitzt
- Zwei Aktionen: *Langsam*, *Schnell*
- Wer schneller fährt, wird doppelt belohnt



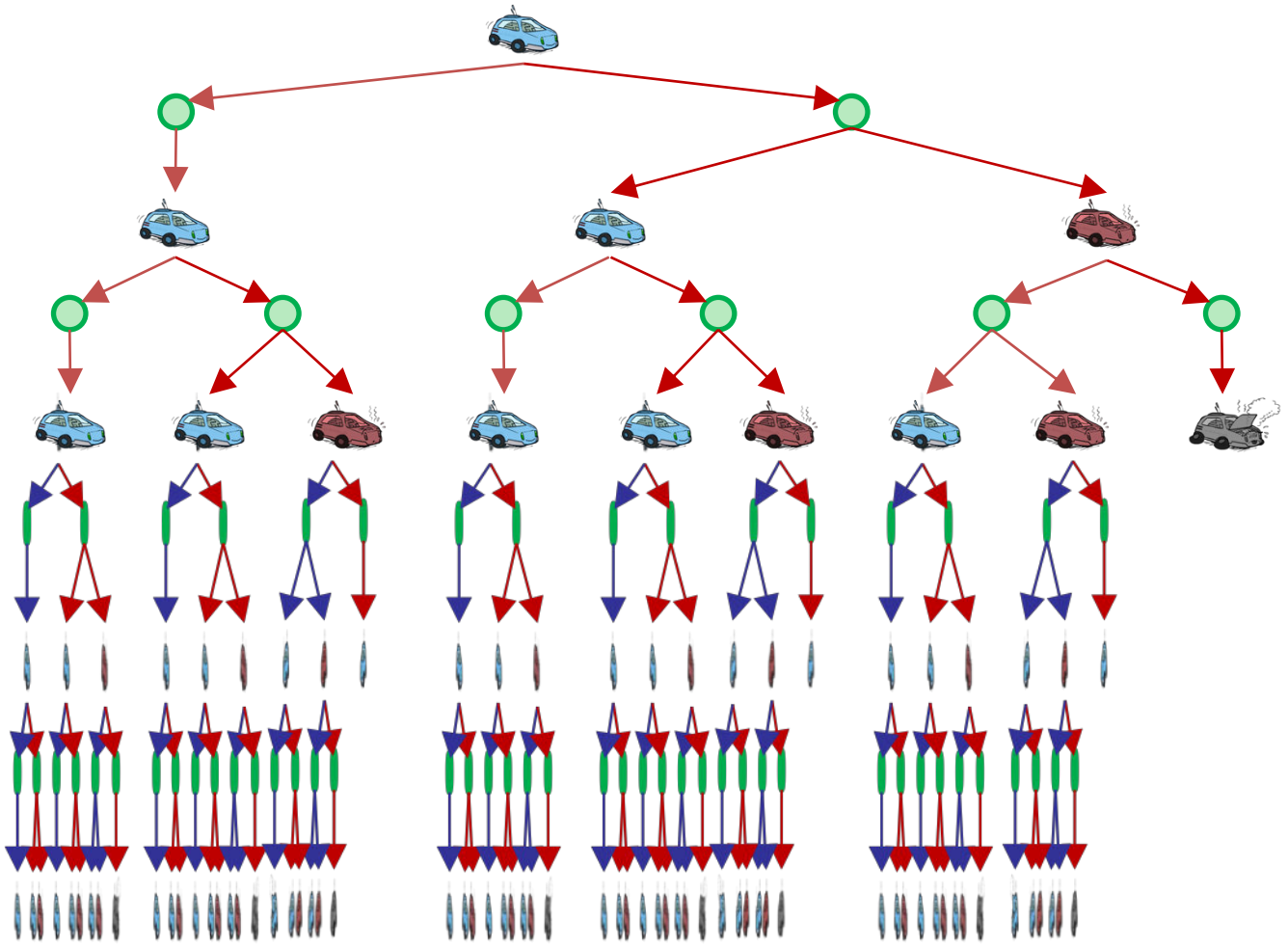
Auto-Rennen Suchbaum

Suchbaum der Tiefe 2:



Auto-Rennen Suchbaum

- Wir machen viel zu viel Arbeit mit expectimax!
- **Problem:** Zustände/States werden wiederholt
- **Problem:** Der Baum wird immer tiefer



Ziel des Agenten

Ziel: Suche nach einer Policy, die den erwarteten kumulativen zukünftigen Reward (auch **Return** genannt) maximiert

$$J_t = \mathbb{E}_\pi \Big[\underbrace{\sum_{k=0}^{\infty} \gamma^k r_{t+k}}_{\text{return } G_t} \Big]$$

- Discount-Faktor: $0 \leq \gamma < 1$
- Trade-off zwischen der Optimierung der langfristigen ($\gamma \to 1$) und kurzfristigen ($\gamma \to 0$) Rewards

Quiz: Discounting

- **Gegeben:**

  - Aktionen: Ost, West und Ausgang (nur in den Ausgangszuständen a, e verfügbar)
  - Übergänge: deterministisch

- **Quiz 1:** Was ist die optimale Policy für $\gamma = 1$?

- **Quiz 2:** Was ist die optimale Policy für $\gamma = 0{,}1$?

- **Quiz 3:** Für welche γ sind West und Ost gleich gut, wenn sie im Zustand d sind?

Optimalitätsdefinition

Der (optimale) Value eines Zustands s:

V^* (s) = erwartete summierter Reward wenn Agent in s started und optimal handelt

$$V^*(\boldsymbol{s}) = \max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| \boldsymbol{s}_0 = \boldsymbol{s} \right]$$

Der (optimale) Value eines Zustands-Aktionspaares (s,a):

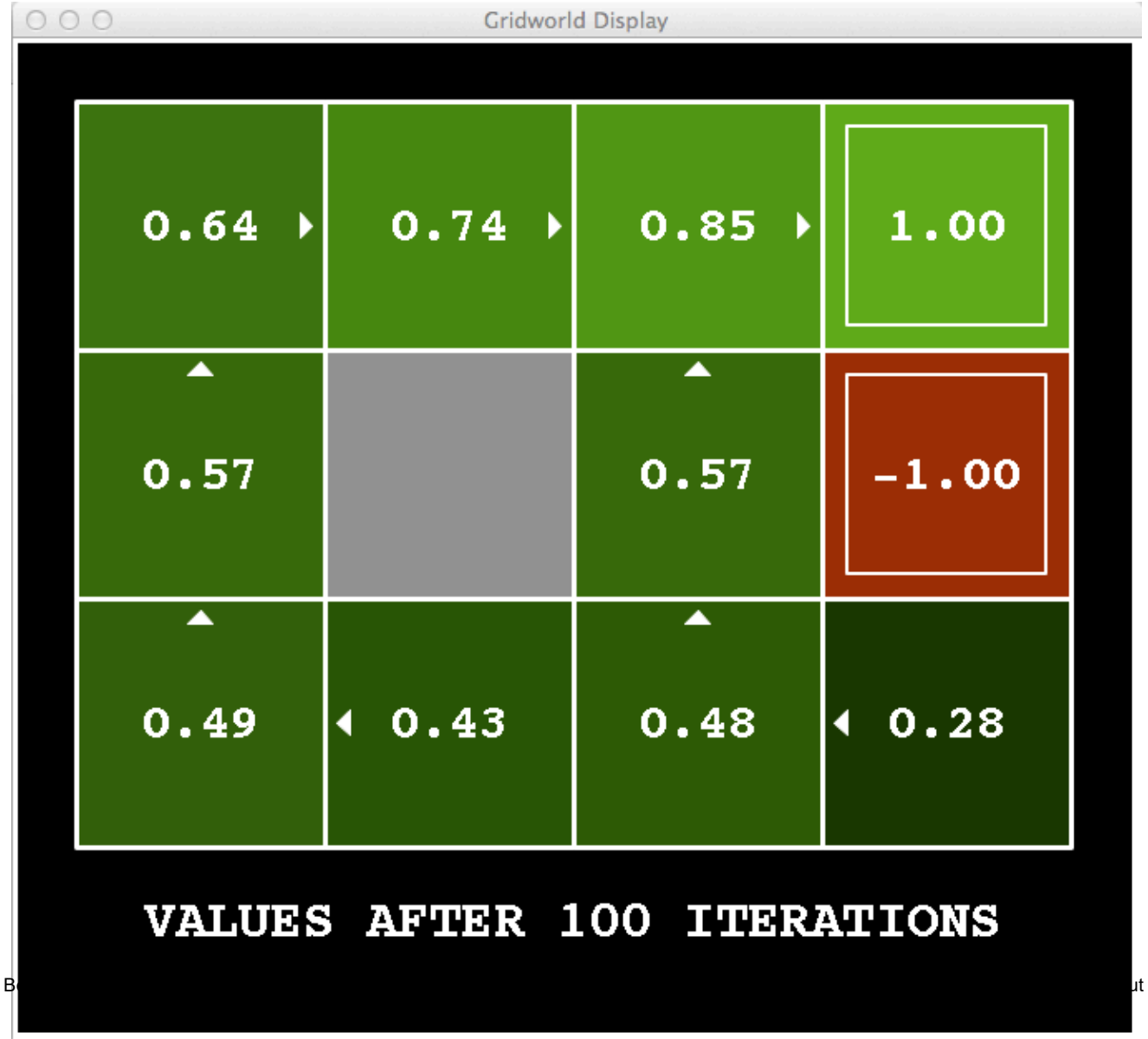
Q^* (s,a) = erwartete summierte Reward, wenn man vom Zustand s aus die Aktion a ausführt und (danach) optimal handelt

$$Q^*(\boldsymbol{s}, \boldsymbol{a}) = \max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| \boldsymbol{s}_0 = \boldsymbol{s}, \boldsymbol{a}_0 = \boldsymbol{a} \right]$$

Die optimale Policy:

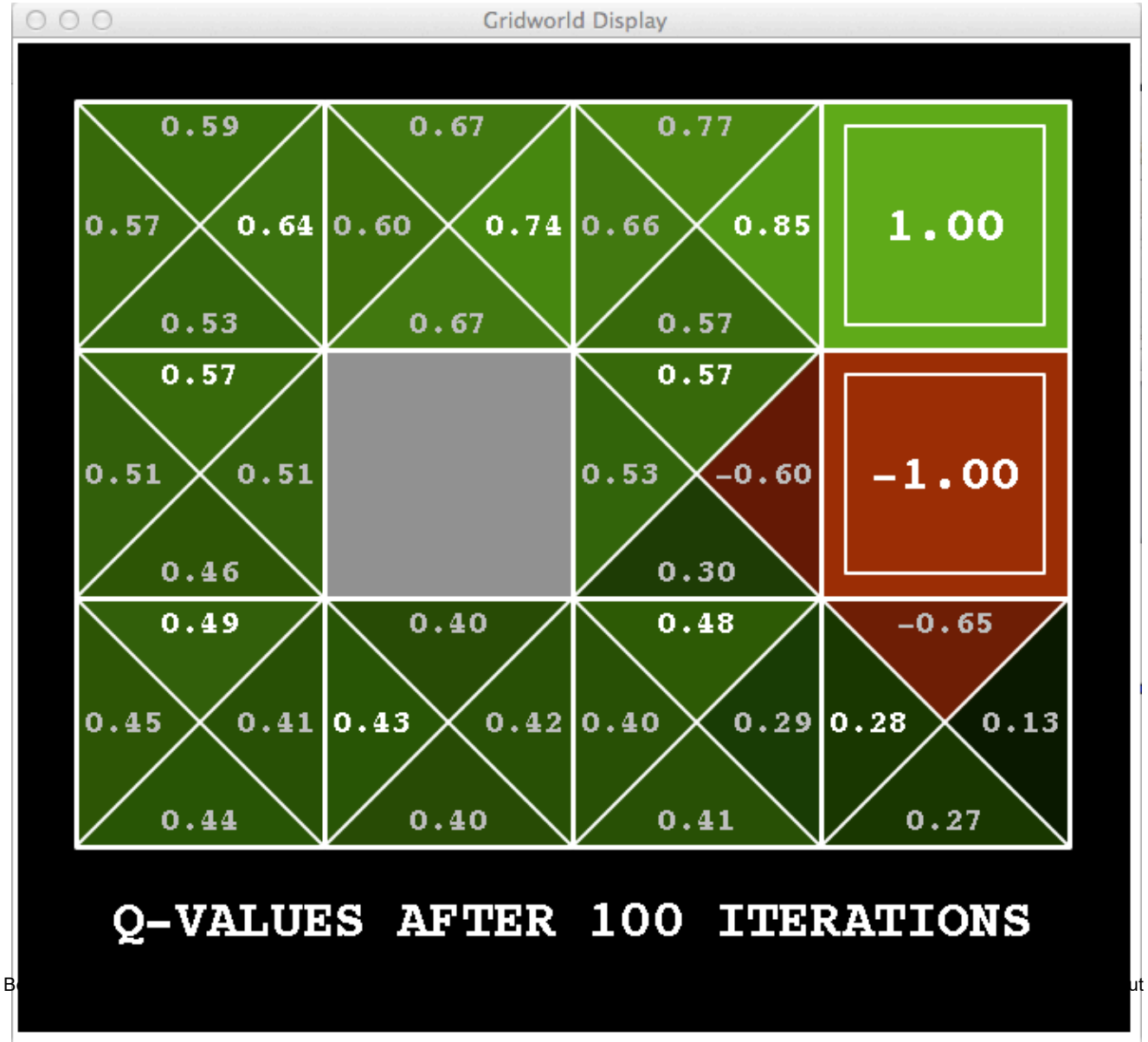
π^*(s) = optimale Aktion im Zustand s

Gridworld V-Values



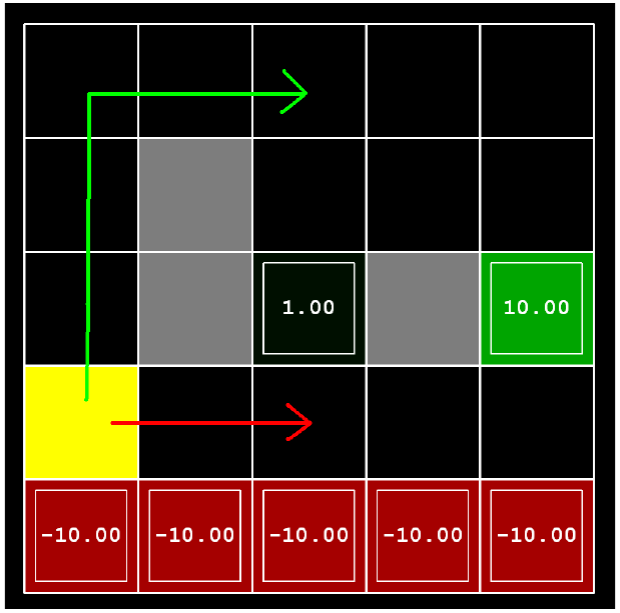
Bewegungsunsicherheit = 0,2
Discount = 0,9
Living Reward = 0

Gridworld Q-Values



Bewegungsunsicherheit = 0,2
Discount = 0,9
Living Reward = 0

Ein Beispiel...



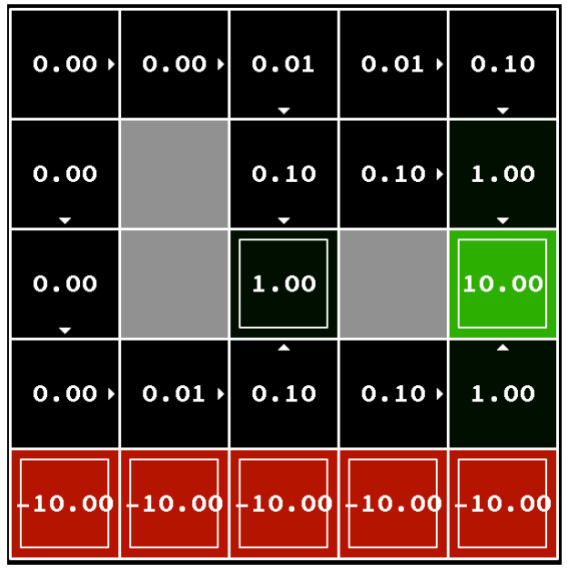
Welche Lösungen werden bevorzugt?

a) Naher Ausgang (+1), riskiere die Klippe (-10)
b) Naher Ausgang (+1), aber vermeiden Sie die Klippe (-10)
c) Entfernter Ausgang (+10), riskiere die Klippe (-10)
d) Entfernter Ausgang (+10), vermeiden Sie die Klippe (-10)

... gehören zu welchem Setting?

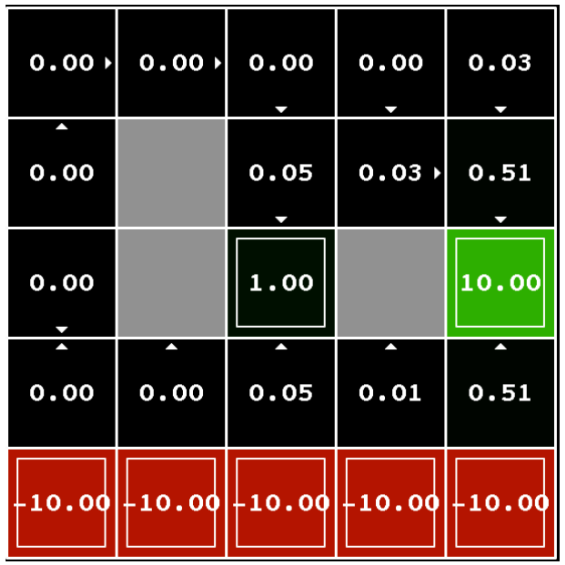
1) $\gamma = 0{,}1$, Bewegungsunsicherheit = 0,5
2) $\gamma = 0{,}99$, Bewegungsunsicherheit = 0
3) $\gamma = 0{,}99$, Bewegungsunsicherheit = 0,5
4) $\gamma = 0{,}1$, Bewegungsunsicherheit = 0

Lösungen



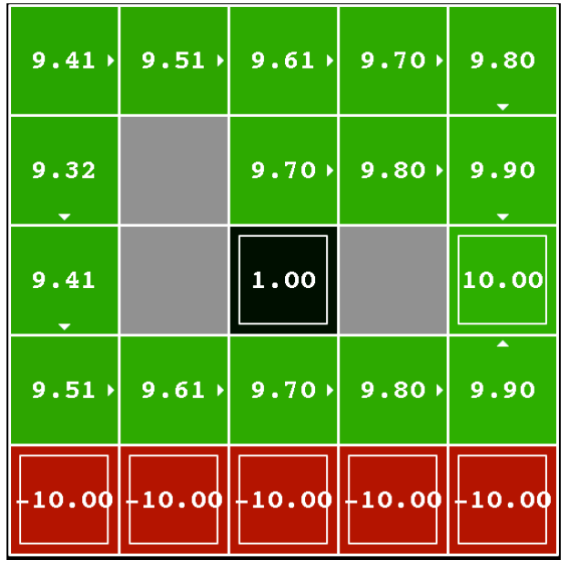
(4) γ = 0,1, Bewegungsunsicherheit = 0

(a) Bevorzuge Ausgang (+1), riskieren Sie die Klippe (-10)



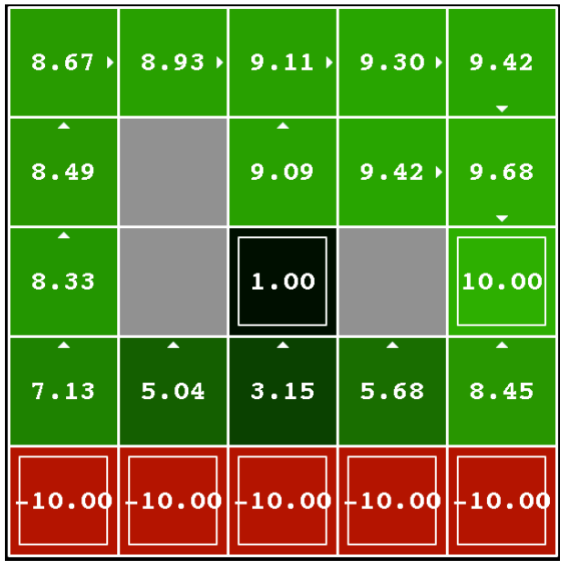
(1) γ = 0,1, Bewegungsunsicherheit = 0,5

(b) Bevorzuge nahen Ausgang (+1), vermeiden Sie die Klippe (-10)



(2) γ = 0,99, Bewegungsunsicherheit = 0

(c) Bevorzuge entfernten Ausgang (+10), riskieren Sie die Klippe (-10)



(3) γ = 0,99, Bewegungsunsicherheit = 0,5

(d) Bevorzuge entfernten Ausgang (+10), vermeiden Sie die Klippe (-10)

Lösen von MDPs - Value-Iteration

Auf den Schultern von Giganten...

"Eine optimale Folge von Aktionen in einem mehrstufigen Optimierungsproblem hat die Eigenschaft, dass **unabhängig von der anfänglichen Stufe, dem Zustand und der Aktionen** die **verbleibenden Steuerungen eine optimale Folge von Entscheidungen für das verbleibende Problem** darstellen müssen, wobei die Stufe und der Zustand, die sich aus den vorhergehenden Steuerungen ergeben, als Anfangsbedingungen betrachtet werden".

Richard Bellman, Dynamische Programmierung, 1957

Values of States

Um den optimalen Value im Zustand s zu erreichen, müssen wir auch in den Folgezuständen optimal handeln

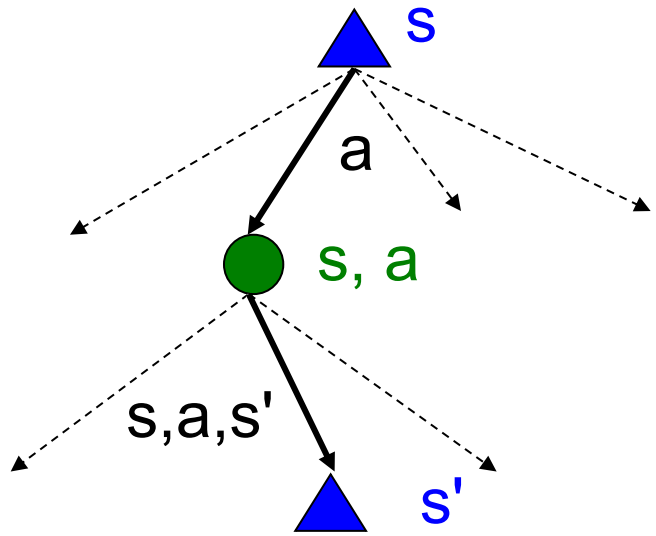
1. Bei Zustand s optimal handeln (Maximierungsschritt)

$$V^*(s) = \max_a Q^*(s,a)$$

2. Erwartungswert über den Value des nächsten Z

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s')$$

Im Zustand s' optimal handeln



Bellman'sches Optimalitätsprinzip: Rekursive Definition von optimalen Values

$$V^*(s) = \max_a \left(r(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s') \right)$$

Aber wie können diese Values berechnet werden?

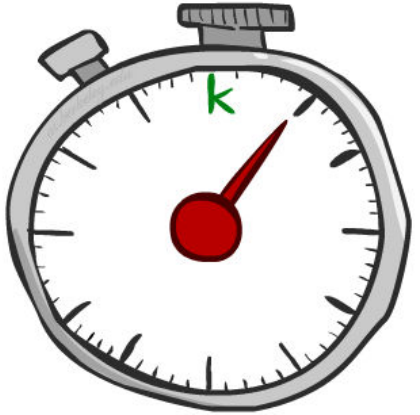
Valuefunktionen mit begrenzten Horizont

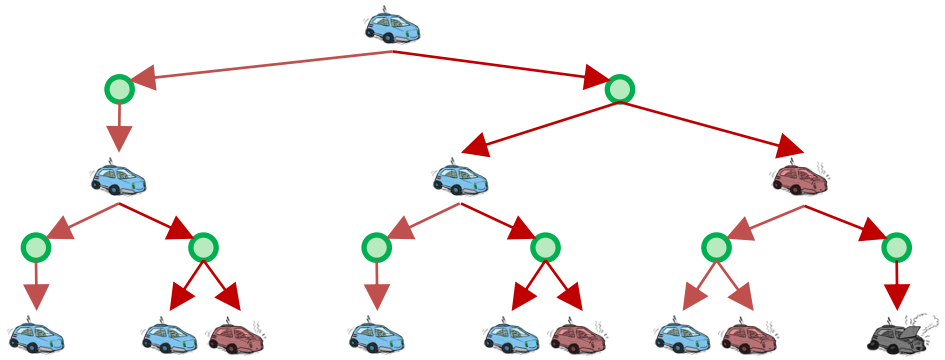
Definiere V_k (s) als den optimalen Value von s, wenn der MDP in k Zeitschritten endet

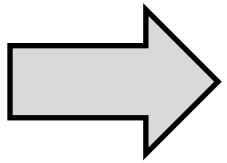
- Äquivalent: Expectimax der Tiefe k von s startend
- Kann effizient aus V_{k-1} (s) berechnet werden.

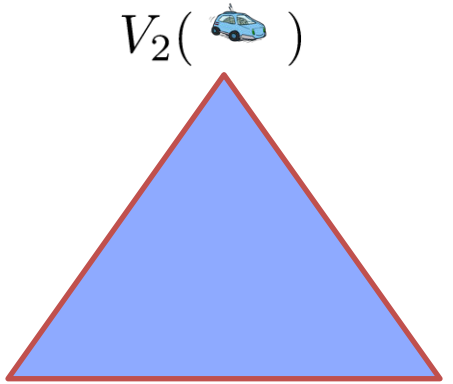
$$V_k^*(s) = \max_a \left(r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_{k-1}^*(s') \right)$$

- Wie ein 1-Schritt-Expectimax mit Endwerten V_{k-1} (s)

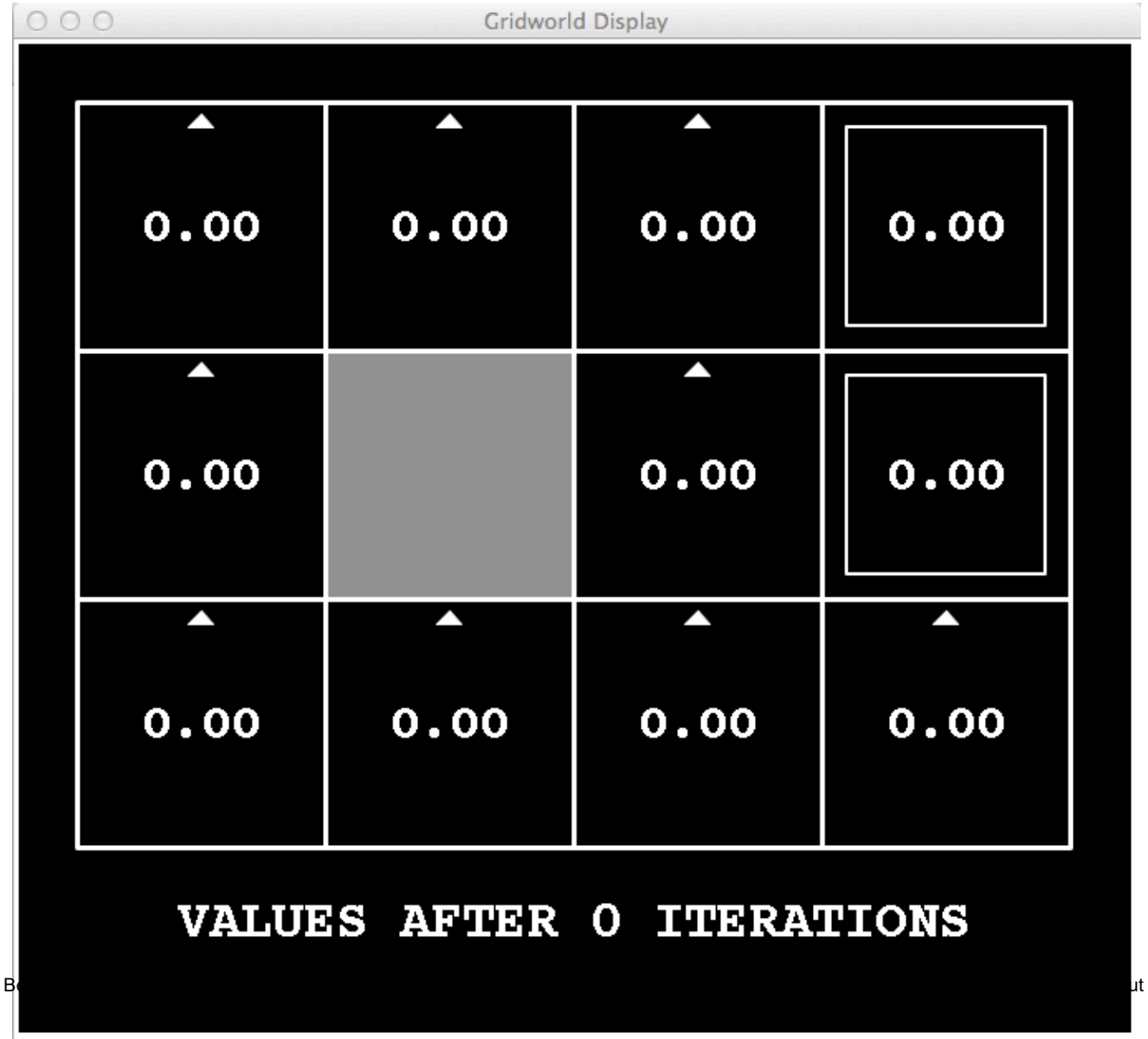






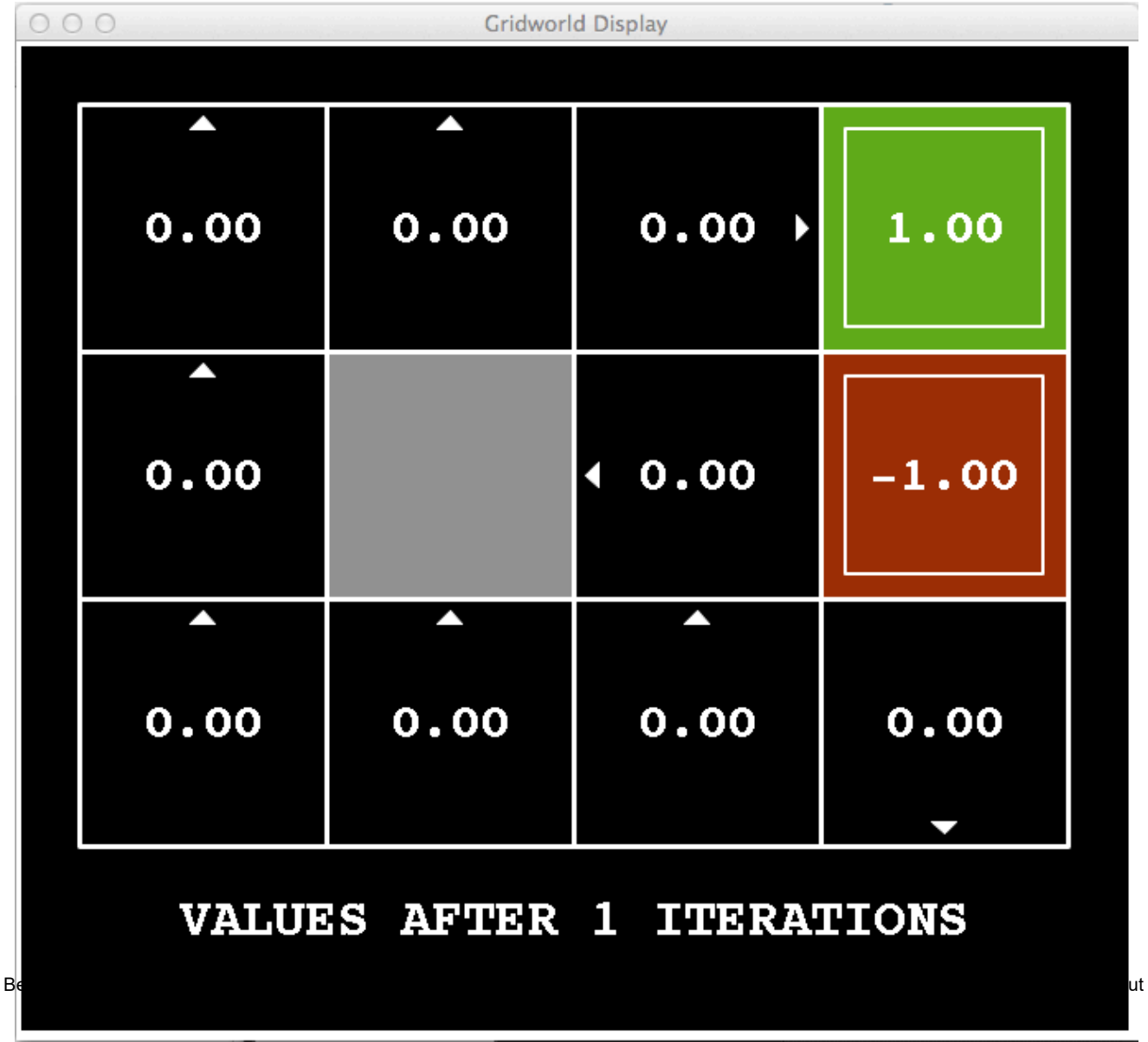


k=0



Bewegungsunsicherheit = 0,2
Discount = 0,9
Living Reward = 0

k=1



Bewegungsunsicherheit = 0,2
Discount = 0,9
Living Reward = 0

k=2

Bewegungsunsicherheit = 0,2
Discount = 0,9
Living Reward = 0

k=3

Bewegungsunsicherheit = 0,2
Discount = 0,9
Living Reward = 0

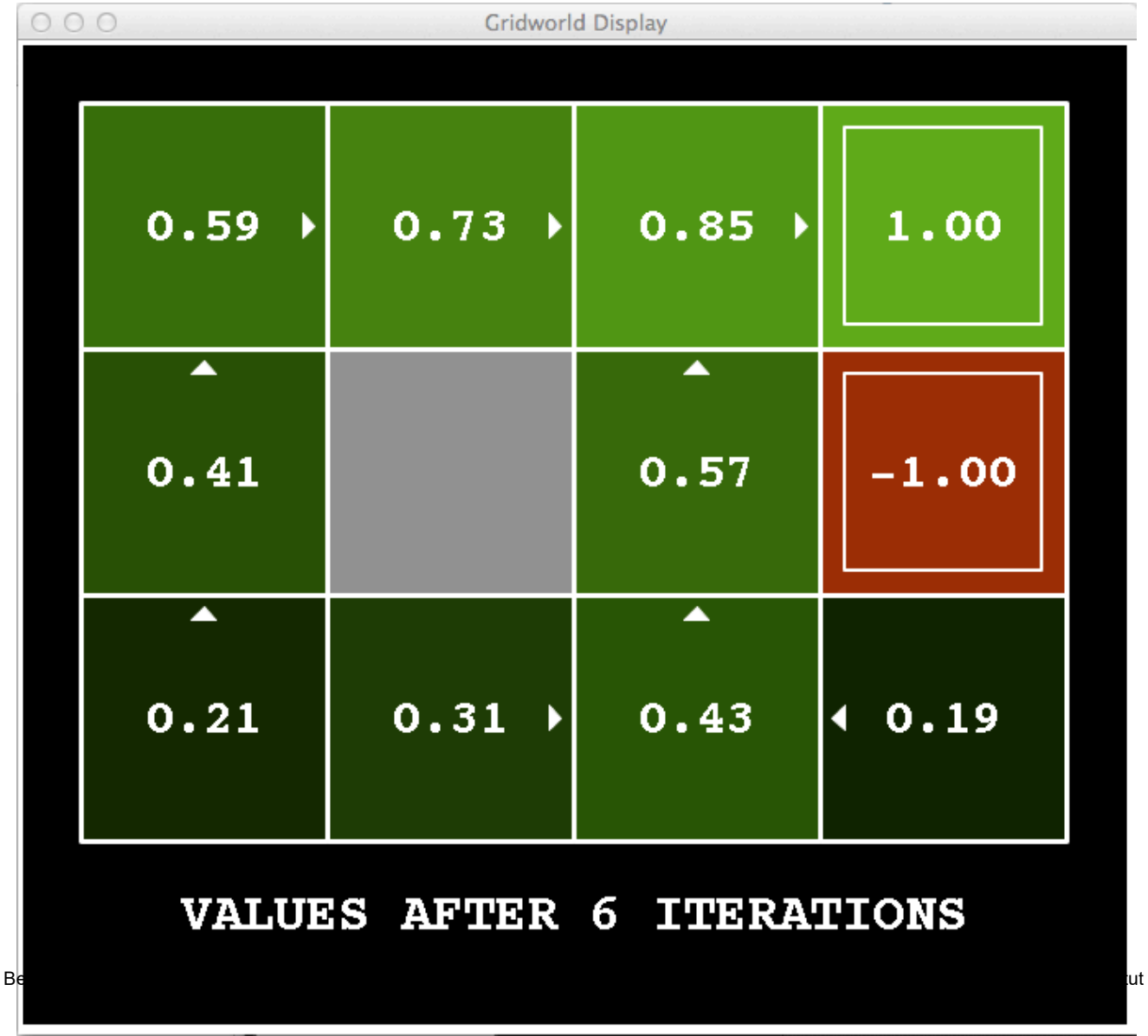
k=4

Bewegungsunsicherheit = 0,2
Discount = 0,9
Living Reward = 0

k=5

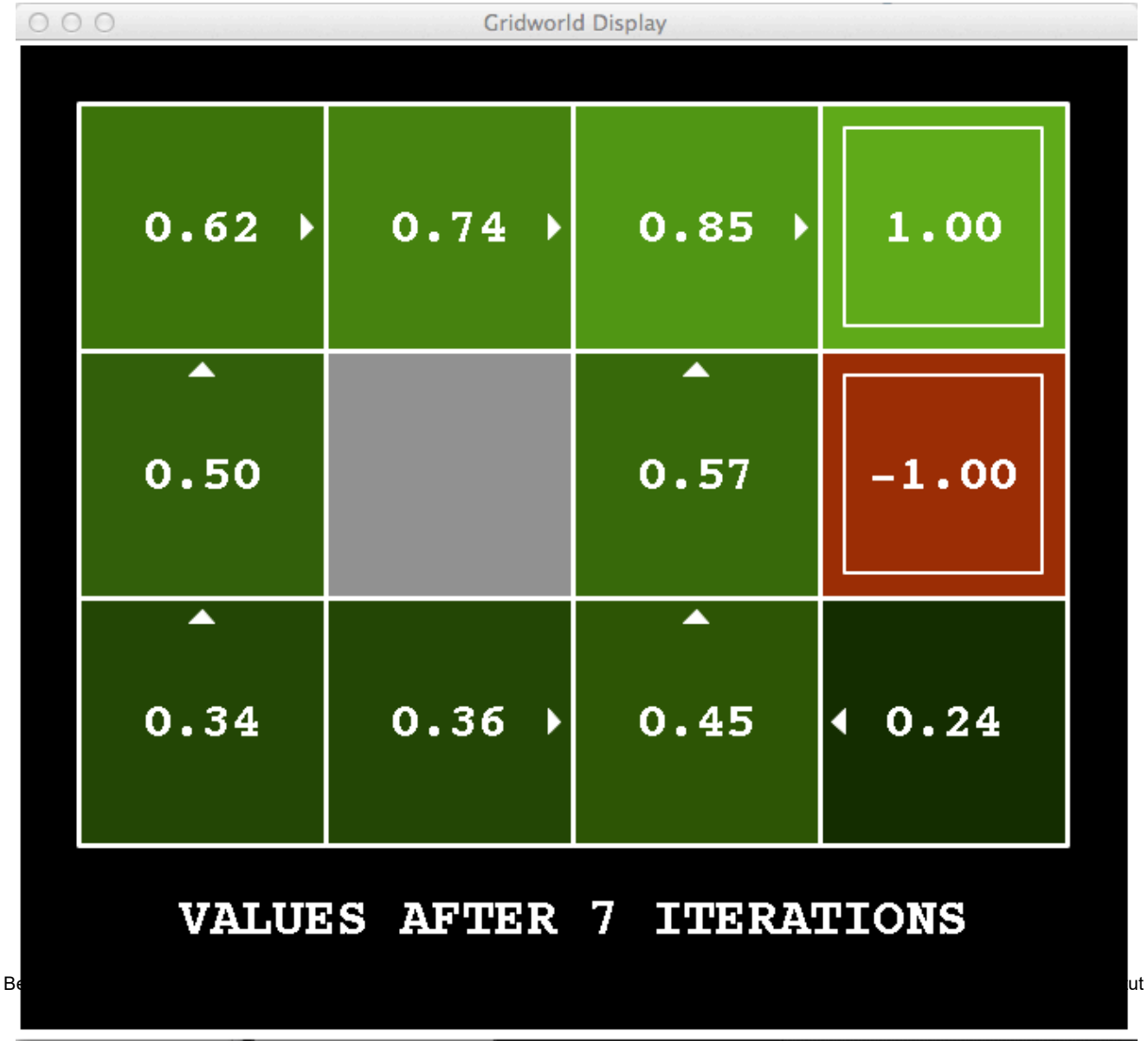
Bewegungsunsicherheit = 0,2
Discount = 0,9
Living Reward = 0

k=6



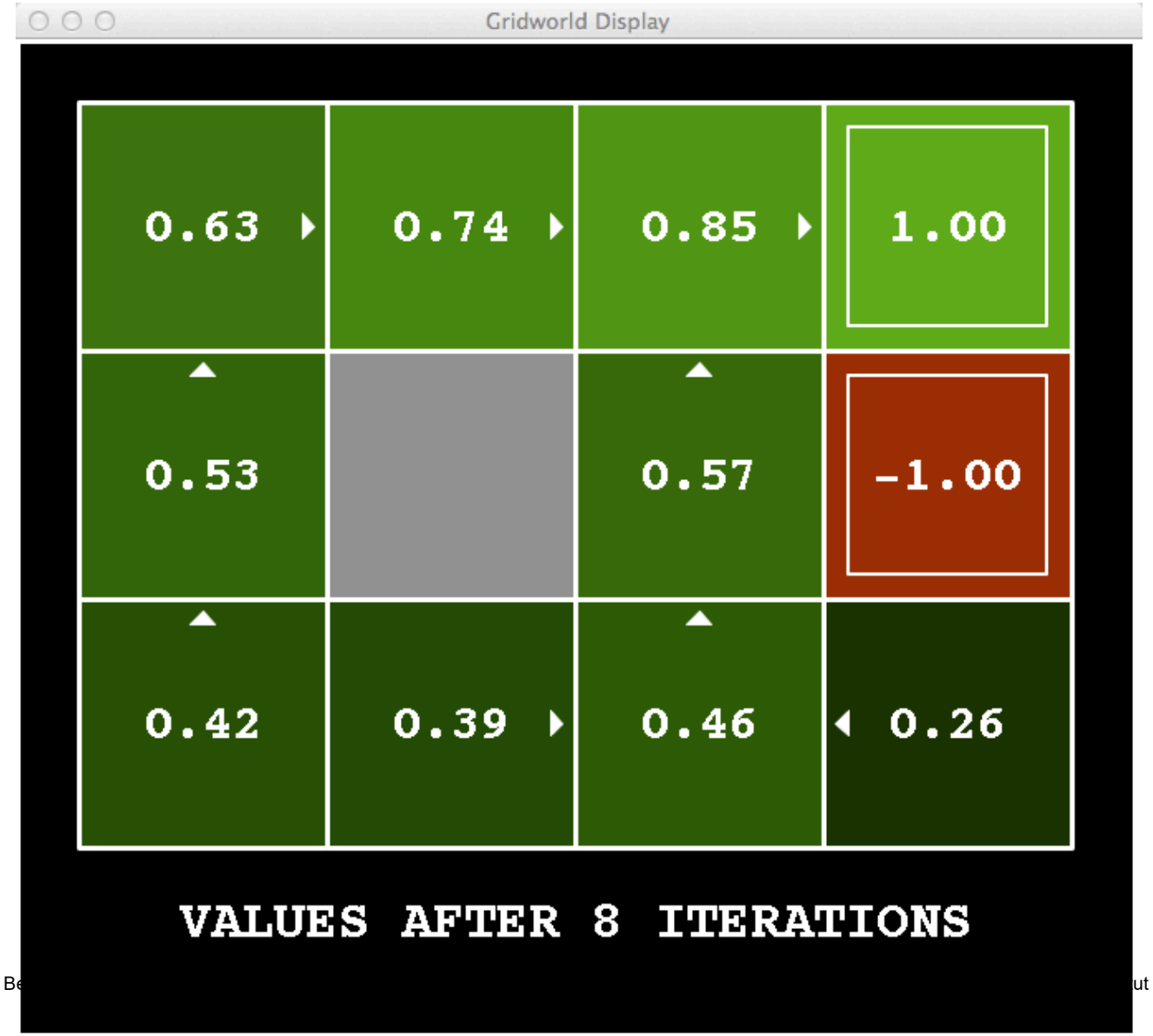
Bewegungsunsicherheit = 0,2
Discount = 0,9
Living Reward = 0

k=7



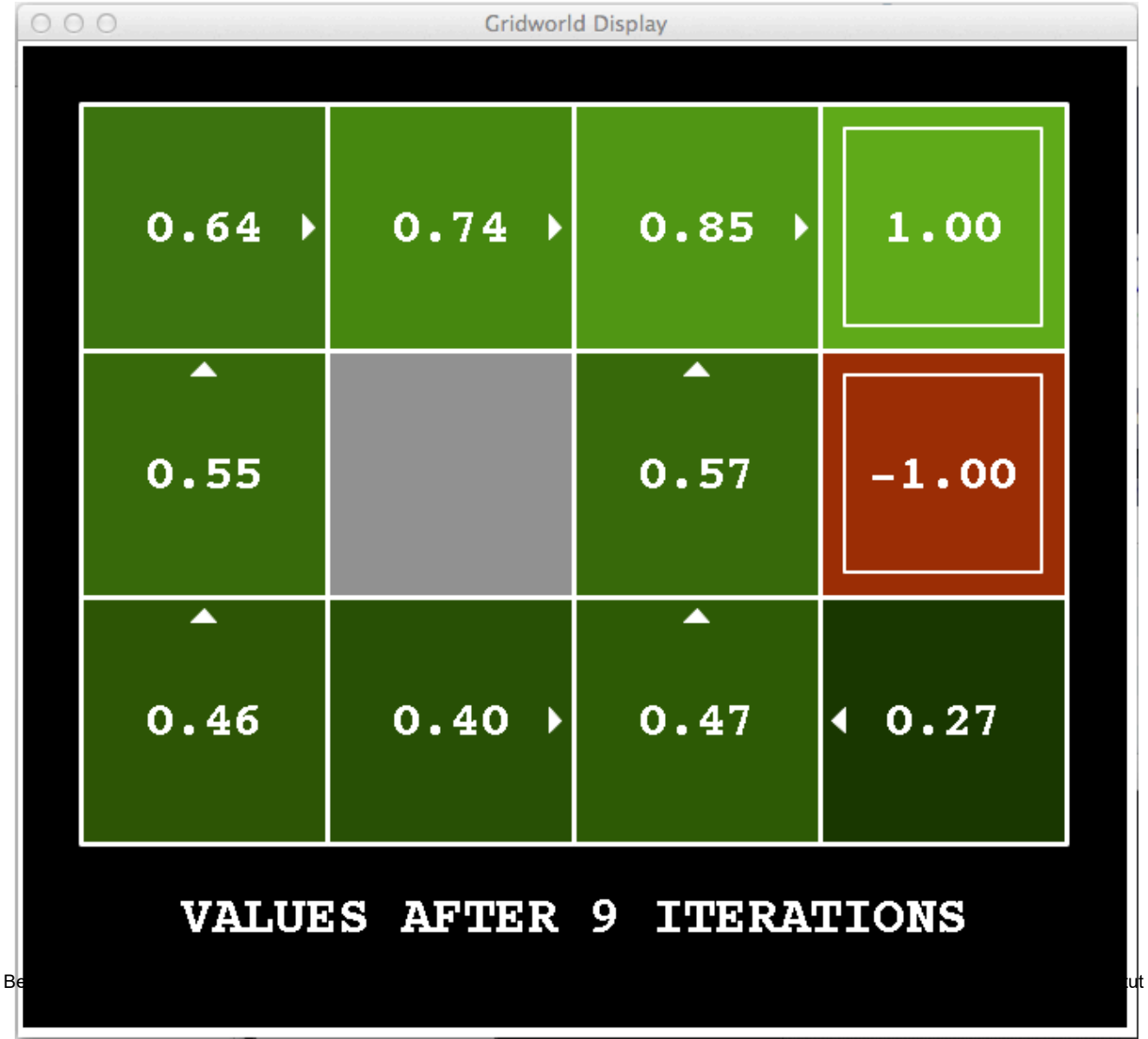
Bewegungsunsicherheit = 0,2
Discount = 0,9
Living Reward = 0

k=8



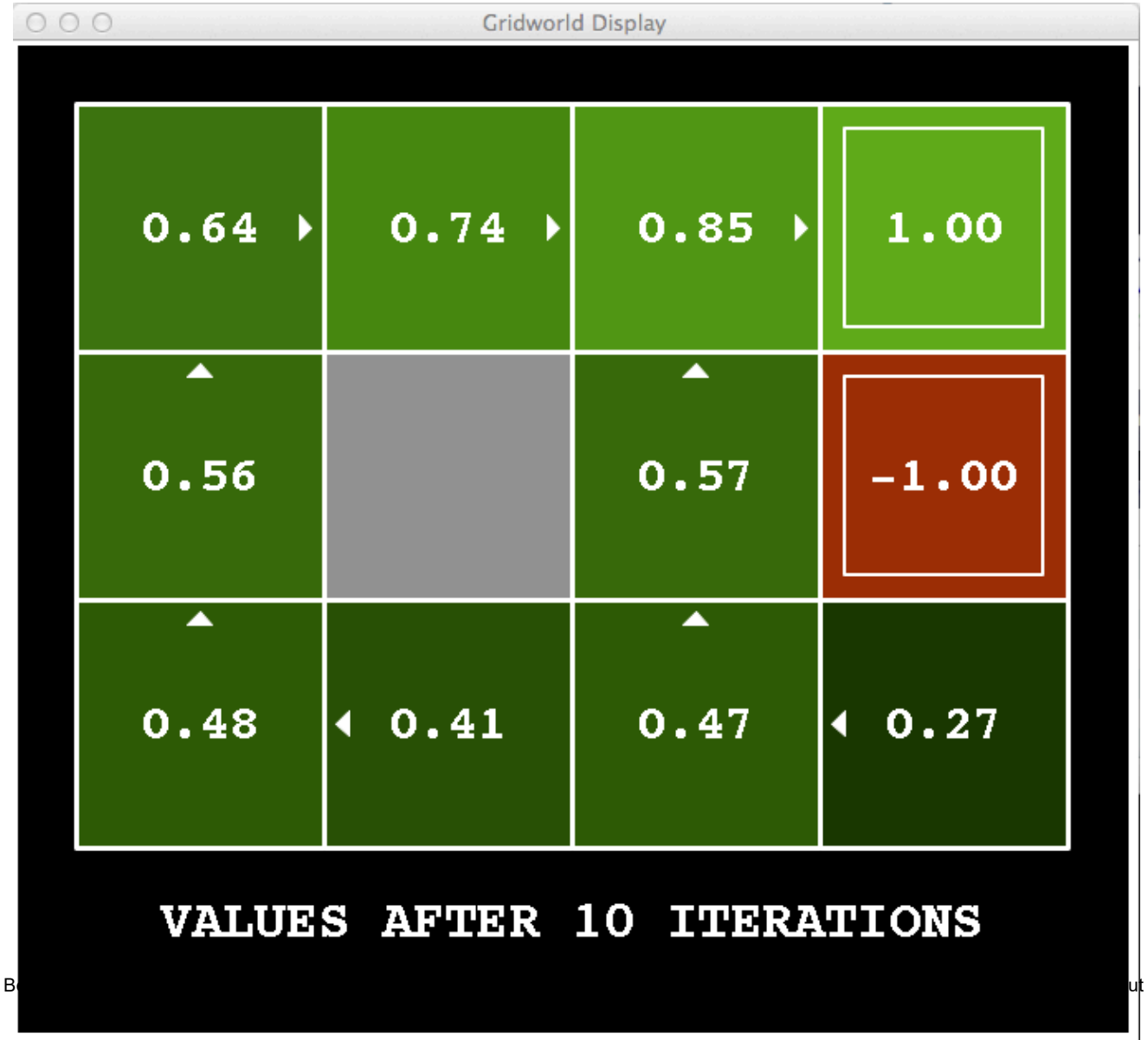
Bewegungsunsicherheit = 0,2
Discount = 0,9
Living Reward = 0

k=9



Bewegungsunsicherheit = 0,2
Discount = 0,9
Living Reward = 0

k=10

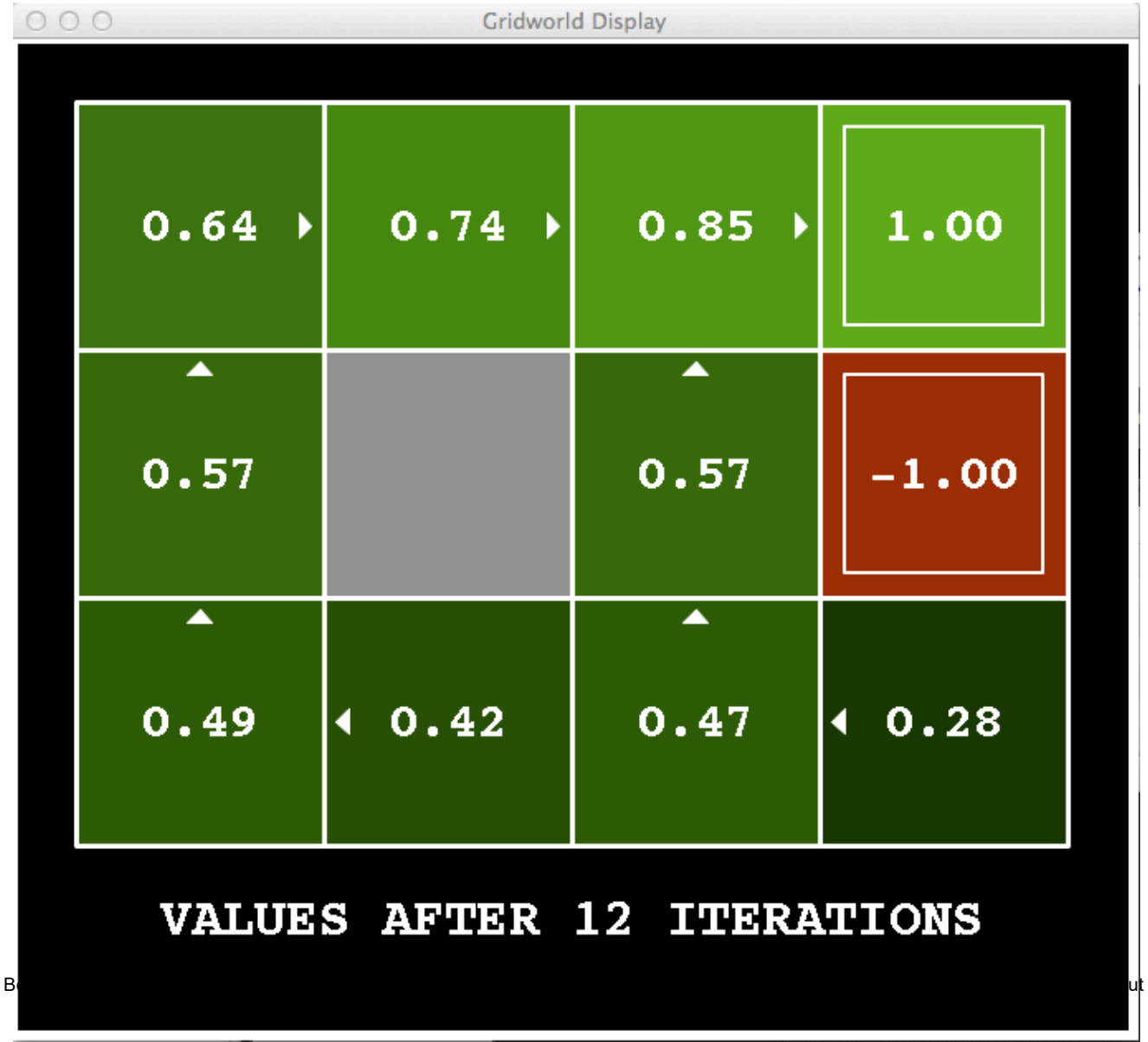


Bewegungsunsicherheit = 0,2
Discount = 0,9
Living Reward = 0

k=11

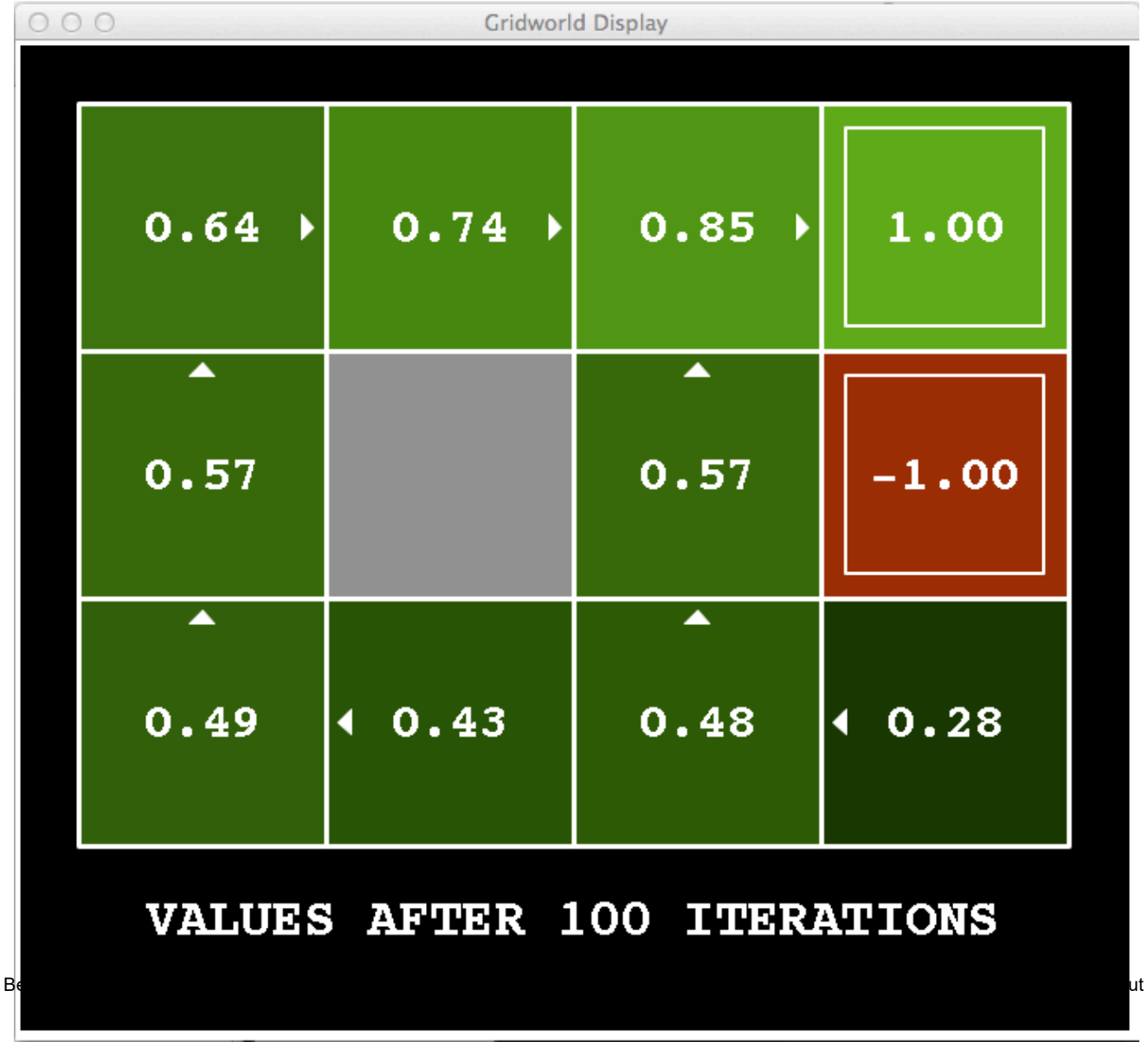
Bewegungsunsicherheit = 0,2
Discount = 0,9
Living Reward = 0

k=12



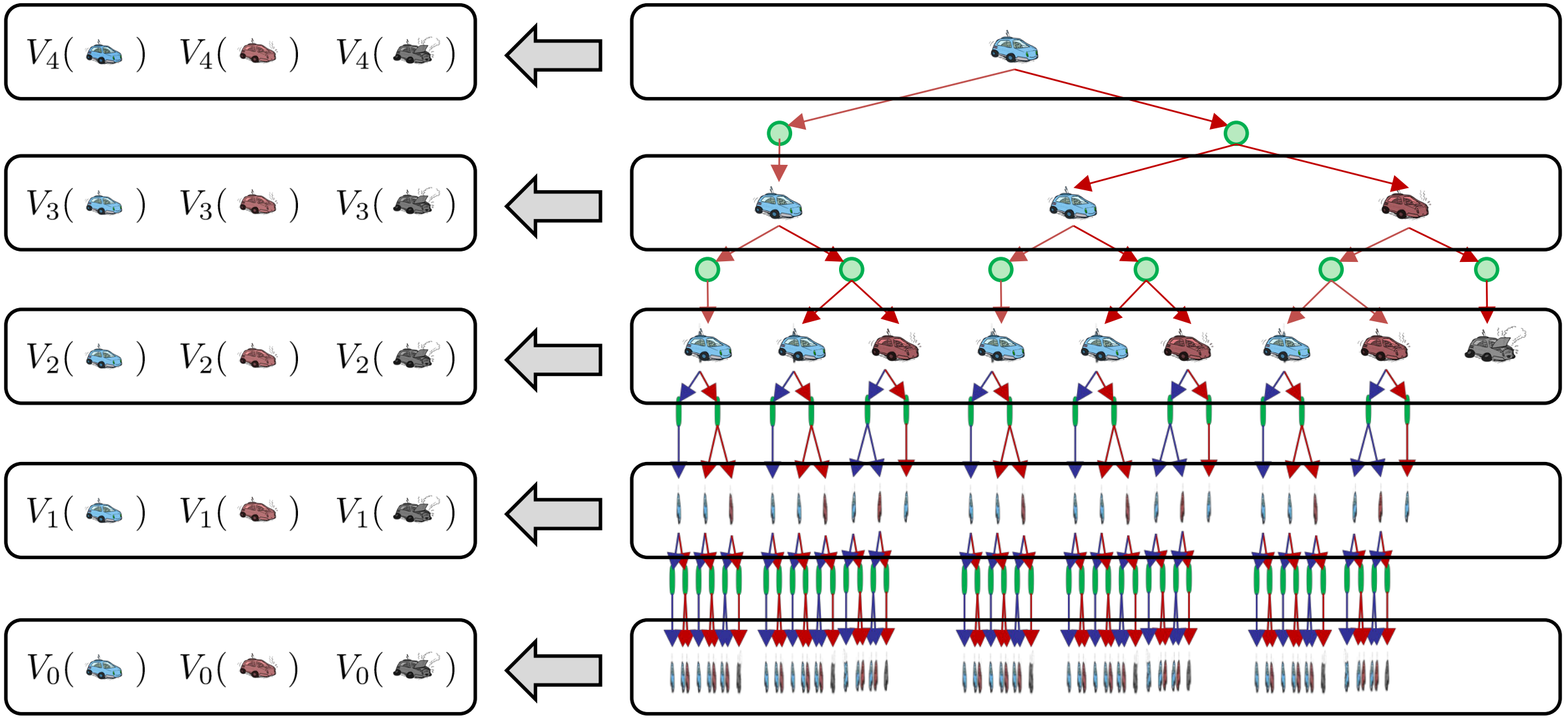
Bewegungsunsicherheit = 0,2
Discount = 0,9
Living Reward = 0

k=100



Bewegungsunsicherheit = 0,2
Discount = 0,9
Living Reward = 0

Berechnung zeitlich begrenzter Values



Value-Iteration

Value-Iteration Algorithmus

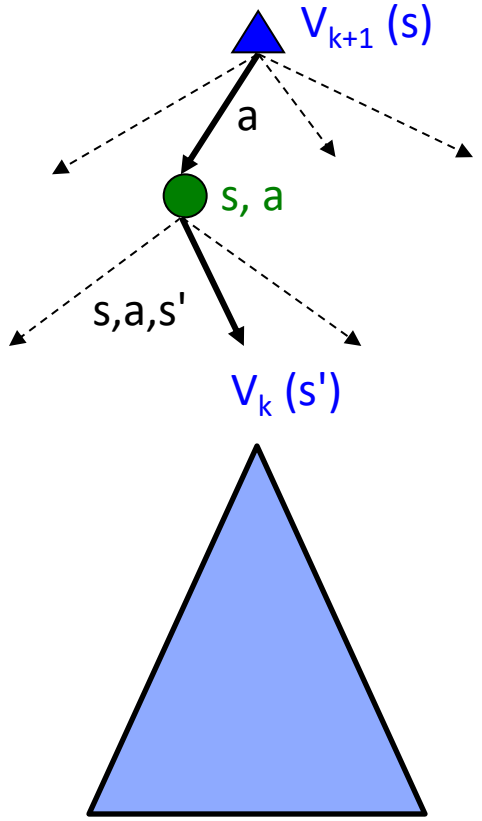
Init: $V_0^\pi(s) = 0$

Repeat

Bei gegebenem Vektor der Values V_{k-1} (s), Berechnung von V_k (s) (für alle Zustände):

$$V_k^*(s) = \max_a \left(r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_{k-1}^*(s') \right)$$

Until Convergence



Komplexität der einzelnen Iterationen: $O(S^2 A)$

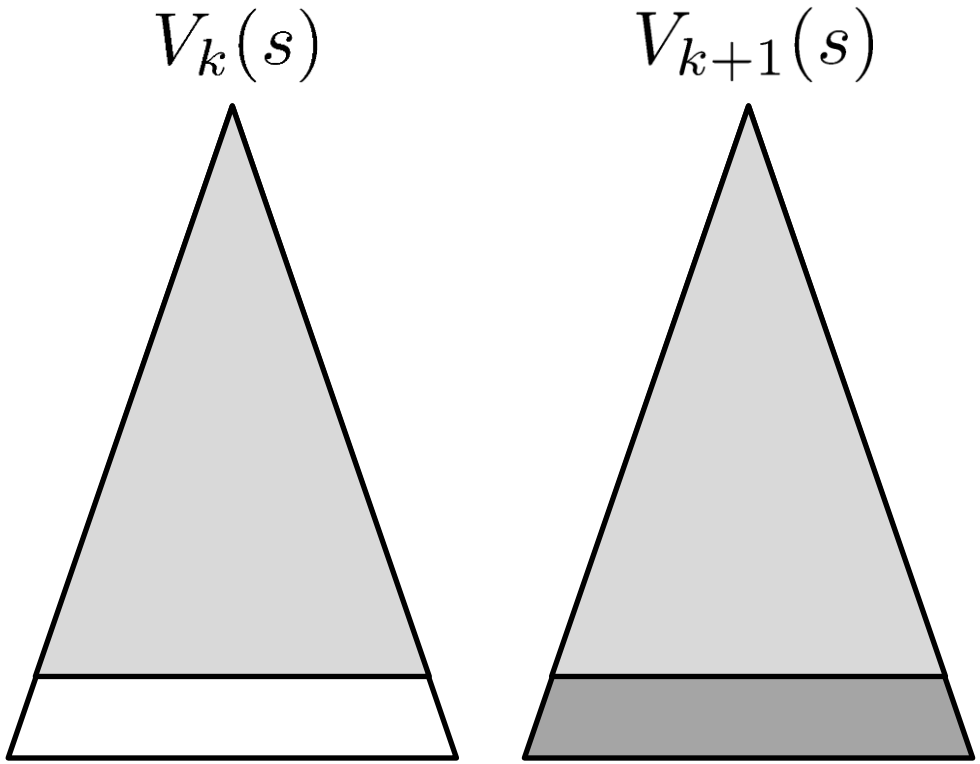
Konvergenz? (Selbstudium)

Theorem: Value-Iteration konvergiert. Bei Konvergenz haben wir die optimale Valuefunktion V* für das Problem des discounted unendlichen Horizonts gefunden, die die Bellman-Gleichungen erfüllt

$$\forall s \in S: \; V^*(s) = \max_a \left(r(s,a) + \gamma \sum_{s'} p(s'|s,a) V^*(s') \right)$$

Skizze:

- Für jeden Zustand können V_k und V_{k+1} als Tiefe k+1 expectimax Ergebnisse in nahezu identischen Suchbäumen betrachtet werden
- Der Unterschied besteht darin, dass auf der unteren Ebene V_{k+1} tatsächliche Rewards enthält, während V_k Nullen enthält.
- Diese letzte Schicht ist im besten Fall ganz R_{MAX} (im schlechtesten Fall ist sie R_{MIN})
- Aber alles wird um γ^k discounted
- V_k und V_{k+1} sind also höchstens γ^k max |R| verschieden
- Mit zunehmendem k konvergieren die Values also



Berechnung der optimalen Policy

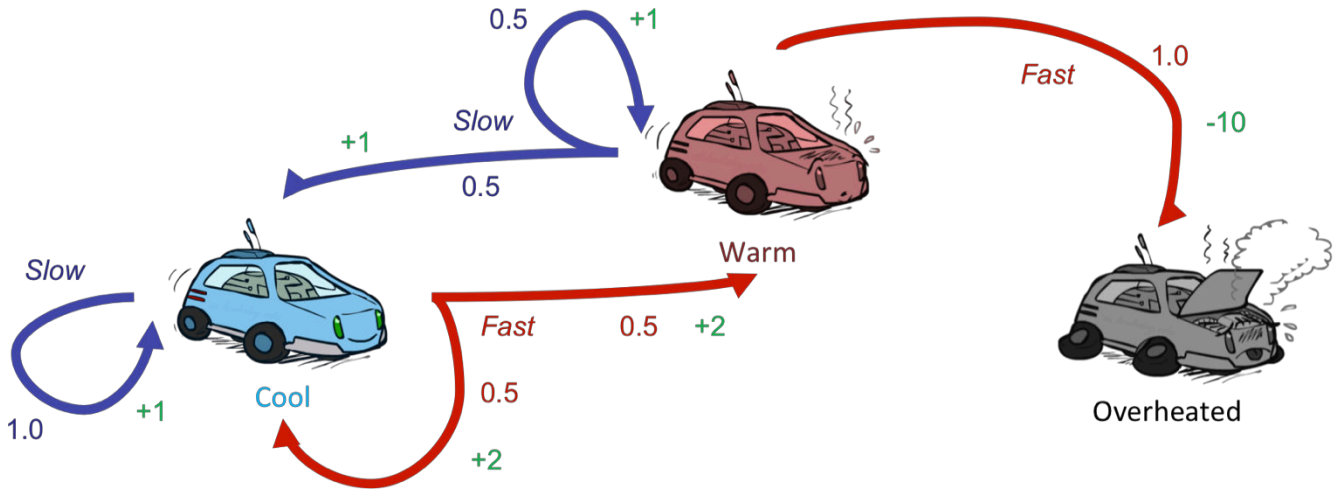
Optimale Policy:

$$\pi^*(\boldsymbol{s}) = \arg\max_{\boldsymbol{a}} \left(r(\boldsymbol{s}, \boldsymbol{a}) + \gamma \sum_{\boldsymbol{s}'} p(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a}) V^*(\boldsymbol{s}') \right)$$

- **Anmerkung:** Die optimale Policy mit unendlichem Horizont ist stationär, d.h. die optimale Aktion im Zustand s ist immer die gleiche und hängt nicht von der Zeit ab. (Effizient zu speichern!)

Beispiel: Wert-Iteration

V_2	3.5	2.5	0
V_1	2	1	0
V_0	0	0	0



Einfachkeitshalber: Discount-factor = 1.0

$$V_k^*(s) = \max_a \left(r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_{k-1}^*(s') \right)$$

Q-Values

Der (optimale) Value eines Zustands-Aktionspaares (s,a):

$$Q^*(s,a) = \left(r(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_{a'} Q^*(s',a') \right)$$

- Q^* (s,a) = erwartete summierte Reward, wenn man vom Zustand s aus die Aktion a ausführt und (danach) optimal handelt

Wir können auch eine Value-Iteration mit den Q-Values durchführen (wichtig für nächste VO):

$$Q_k^*(s,a) = \left(r(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_{a'} Q_{k-1}^*(s',a') \right)$$

- Im Grunde dasselbe wie bei V-Funktionen, nur dass der Max-Operator an einer anderen Stelle steht

Q-Value Iteration

$$Q_k^*(s,a) = \left(r(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_{a'} Q_{k-1}^*(s',a') \right)$$

Die Policy kann direkt von Q berechnet werden:

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

k = 100

Noise = 0.2
Discount = 0.9

Funktioniert das? Ja... aber

Leider können wir dies nur in 2 Fällen tun

- Diskrete Systeme

  ...aber die Welt ist nicht diskret!

- Lineare Systeme, Quadratische Belohnung, Gaußsches Rauschen (LQR)

  ... aber die Welt ist nicht linear!

In allen anderen Fällen müssen wir Näherungswerte verwenden!

- **Darstellung der V-Funktion:** Wie kann V in kontinuierlichen Zustandsräumen dargestellt werden?
- **Wir müssen eine Lösung finden:**

$\max_{\boldsymbol{a}} Q^*(\boldsymbol{s}, \boldsymbol{a})$ Schwierig in kontinuierlichen Aktionsräumen

$\mathbb{E}_{\mathcal{P}}\left[V^*(\boldsymbol{s}') \middle| \boldsymbol{s}, \boldsymbol{a}\right]$: schwierig für beliebige Funktionen V und Modelle

Fragen zum Selbsttest

Was Sie nach diesem Vortrag wissen sollten:

- Die Definition eines MDP
- Warum brauchen wir Discounting?
- Die Definition der optimalen V- und Q-Funktionen
- Was ist das Bellman-Prinzip der Optimalität?
- Wie funktioniert der Value-Iteration Algorithmus?