Kapitel 7
Q-Learning

Mit Lea Reinhart

Wenn Informatikerinnen und Informatiker um Highscores in Videospielklassikern wie Space Invaders, Breakout oder Video Pinball wetteifern, kann das durchaus einen wissenschaftlichen Hintergrund haben. Auch wenn sie probieren, mit einem ferngesteuerten Spielzeugauto Stöcke in vertikaler Balance zu halten, per Quadrokopter einen Hindernisparcours zu durchqueren oder mit einem Roboterarm ein Kartenhaus zu bauen, steckt oft mehr dahinter als nerdiger Spieltrieb.

Die genannten Aufgaben haben gemeinsam, dass sie Gegenstand des sogenannten *verstärkenden Lernens* (engl. *Reinforcement Learning*) sein können. Dieser Begriff bezeichnet eine ganze Klasse von KI-Verfahren, bei denen ein Programm selbstständig lernen kann, durch bloßes Herumprobieren und anhand positiver und negativer Rückmeldungen Aufgaben zu lösen und Ziele zu erreichen.

In diesem Kapitel stellen wir eine prominente Spielart des verstärkenden Lernens vor: das *Q-Learning* mittels *Q-Tabellen*. Wir werden sehen, dass diese Art des Lernens erstaunliche Parallelen aufweist zu der Art und Weise, wie Menschen neue Fertigkeiten erwerben.

Worum es in diesem Kapitel geht



- *Q-Learning* bezeichnet eine Klasse von automatischen Lernverfahren, bei denen ein *Agent* mit einer *Umwelt* interagiert und Lösungsstrategien für eine gegebene Aufgabe nach dem Prinzip »Versuch und Irrtum« findet.
- Wir stellen das Q-Learning mittels *Q-Tabelle* vor, bei dem für jeden Umweltzustand die vielversprechendste Aktion gefunden werden soll.
- Das Beispielprogramm »Q-Lerner« wendet das Verfahren an und macht es auch visuell nachvollziehbar.

Wir steigen in das Thema mit einem anschaulichen Beispiel ein: Stellen Sie sich ein äußerst kurzsichtiges und ziemlich vergessliches Eichhörnchen vor, das einen Garten bewohnt und dort ein paar Nüsse deponiert hat.

Persönliches Exemplar für Pit Noack

7.1 Das Eichhörnchen und das Nussversteck

In Abbildung 7.1 hat das Eichhörnchen Appetit auf eine Nuss. Bedauerlicherweise ist ihm entfallen, wo es die Nüsse versteckt hat. Es beschließt daher, den Garten mit zufälligen Bewegungen zu durchstreifen. Der Garten ist in Zellen eingeteilt, und diese Zellen sind nummeriert – das wird später noch wichtig werden.

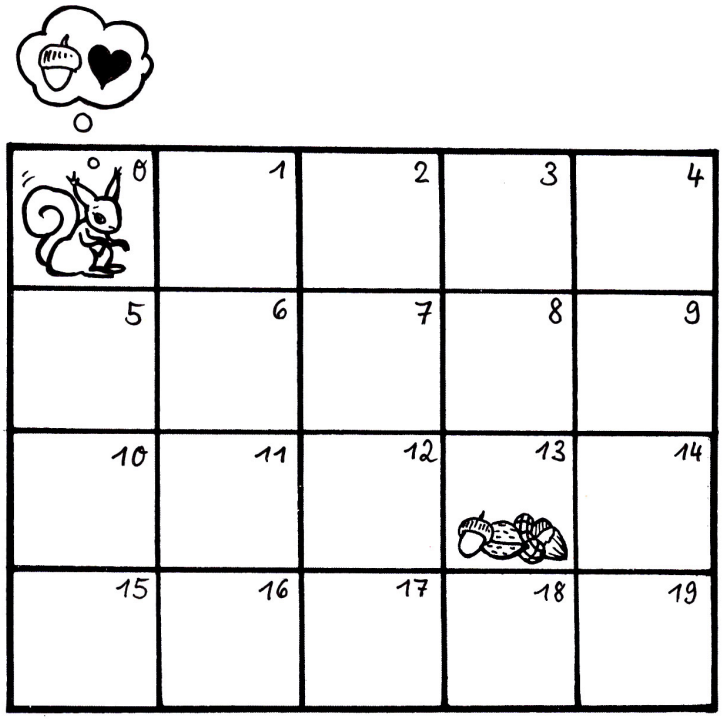


Abbildung 7.1 Das Eichhörnchen hat Appetit, aber leider vergessen, wo die Nüsse versteckt sind.

Schließlich ist das Eichhörnchen auf seinem Zickzackkurs auf das Nussversteck gestoßen. Es hat also sein Ziel erreicht. Im Q-Learning sprechen wir in diesem Fall auch davon, dass eine *Episode* abgeschlossen ist. Das zeigt Abbildung 7.2.

Von diesem Erfolgserlebnis beflügelt, beschließt das Eichhörnchen, eine Merkhilfe anzubringen, die dabei helfen wird, das Nussversteck beim nächsten Mal schneller zu finden. Zu diesem Zweck zeichnet das Eichhörnchen einen Pfeil. Der Pfeil zeigt von der zuletzt besuchten Zelle auf das Nussversteck. Das sehen Sie in Abbildung 7.3.

© Rheinwerk Verlag, Bonn 2023

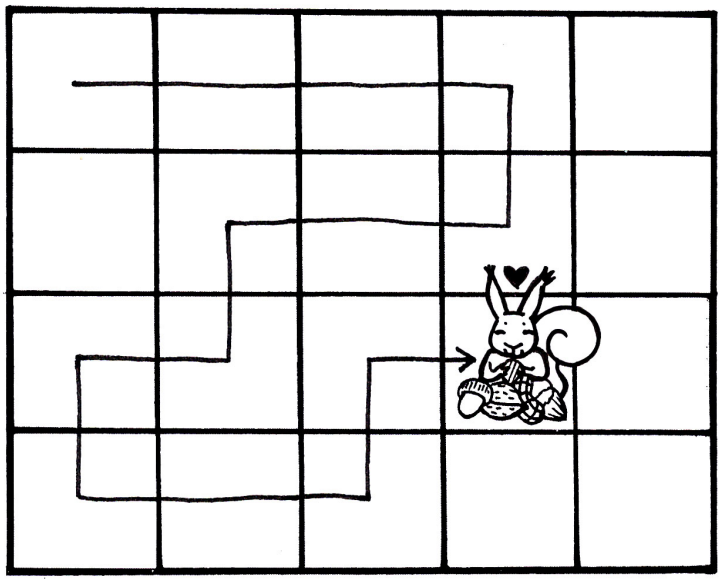


Abbildung 7.2 Durch eine zufällige Suchbewegung hat das Eichhörnchen das Nussversteck gefunden und lässt sich eine Nuss schmecken.

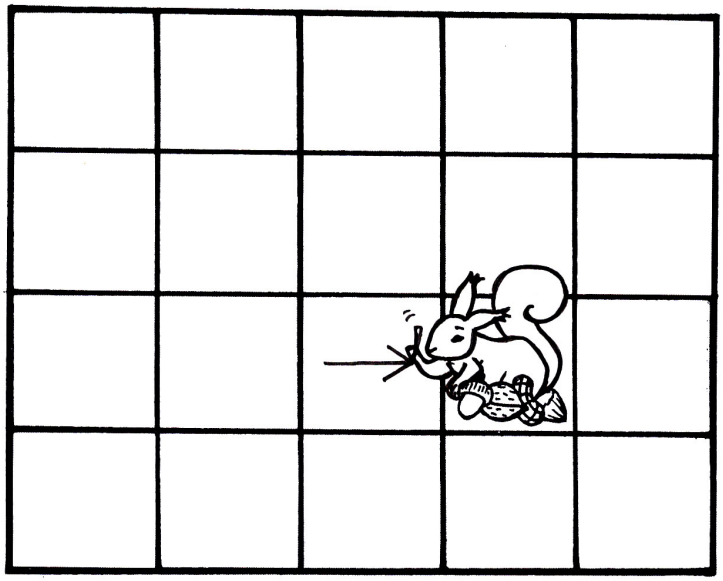


Abbildung 7.3 Das Eichhörnchen zeichnet in die zuletzt betretene Zelle einen Pfeil in Richtung des Nussverstecks.

Persönliches Exemplar für Pit Noack

Wenn das Eichhörnchen beim nächsten Mal Appetit auf eine Nuss verspürt, beginnt eine neue Episode. Da das Eichhörnchen so furchtbar kurzsichtig ist, kann es nicht nach dem Hinweispfeil Ausschau halten. Es muss erneut den Garten mit zufälligen Bewegungen absuchen – und zwar so lange, bis es entweder direkt auf das Nussversteck trifft oder, wie in Abbildung 7.4 gezeigt, auf einen Pfeil.

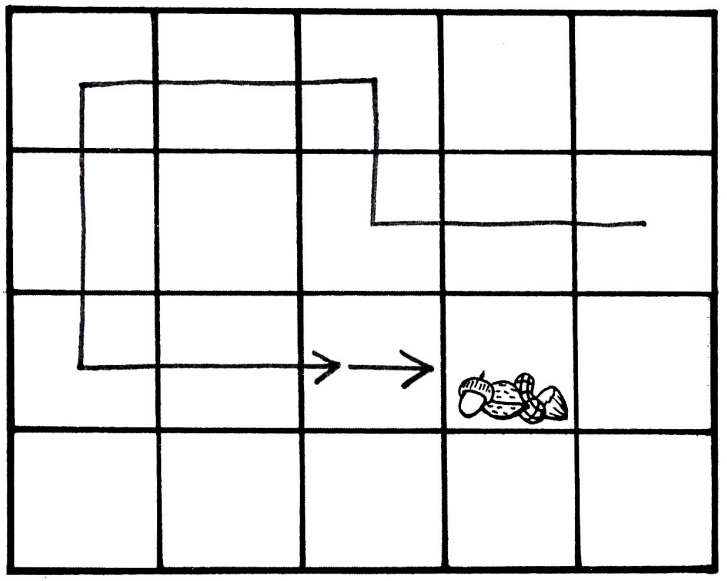


Abbildung 7.4 Jedes Mal, wenn das Eichhörnchen Appetit auf eine Nuss verspürt, startet es wieder eine zufällige Suchbewegung, bis es auf das Nussversteck oder auf einen Pfeil trifft, der zum Nussversteck führt.

Erneut setzt das Eichhörnchen eine Markierung. In Abbildung 7.5 zeichnet es einen Pfeil, der von der zuletzt besuchten Zelle auf den eben gefundenen Pfeil zeigt.

Jedes Mal, wenn das Eichhörnchen nach den Nüssen sucht, also in jeder Episode, wiederholt es dieses Verfahren. Und immer, wenn es einen Pfeil oder das Nussversteck findet, setzt es einen Pfeil von der zuletzt besuchten Zelle in Richtung des gefundenen Pfeils oder des Verstecks.

© Rheinwerk Verlag, Bonn 2023

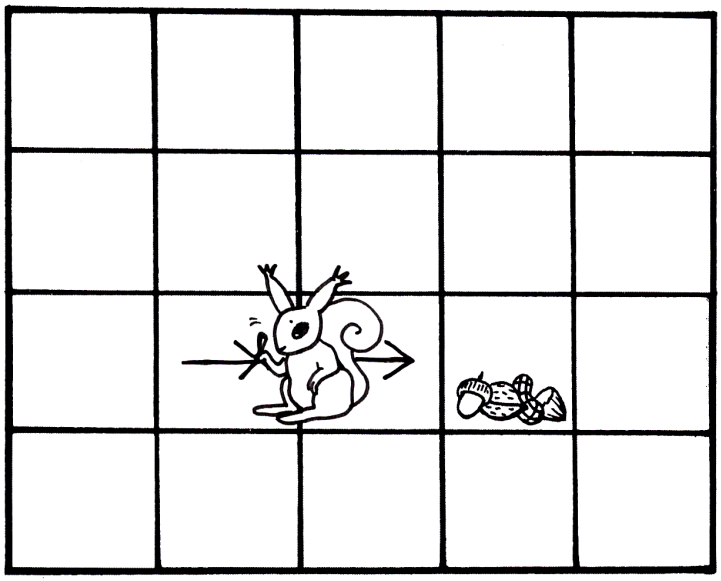


Abbildung 7.5 Das Eichhörnchen zeichnet einen Pfeil in Richtung des gefundenen Pfeils.

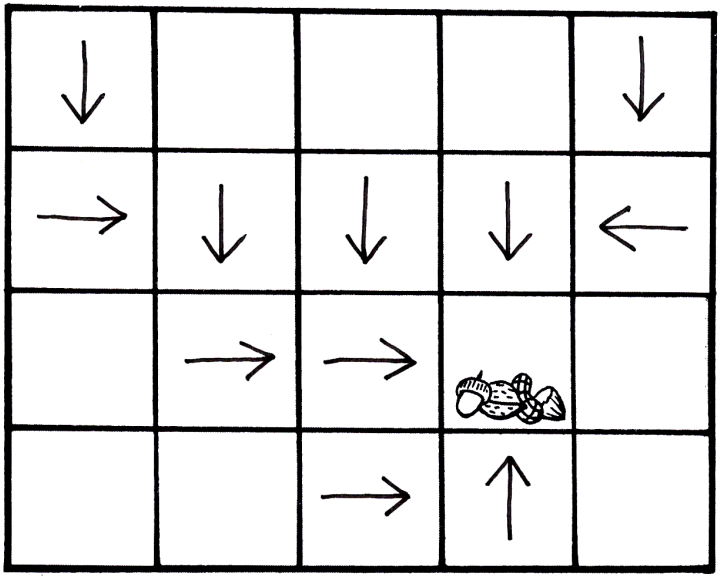


Abbildung 7.6 Immer wenn das Eichhörnchen eine Suche erfolgreich beendet, wird es einen neuen Pfeil hinzufügen ...

Persönliches Exemplar für Pit Noack

Irgendwann hat es sämtliche Zellen mit Pfeilen versehen. Das Eichhörnchen muss nun keine zufälligen Schritte mehr ausführen und kann jedes Mal einen Weg zum Ziel finden, indem es den Pfeilen folgt (siehe Abbildung 7.7).

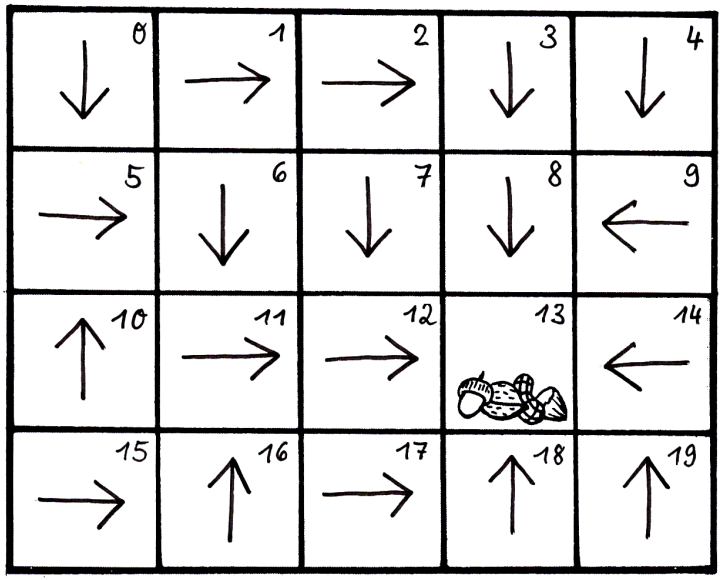


Abbildung 7.7 … bis alle Zellen mit einem Pfeil versehen sind.

Bitte beachten Sie, dass dieses Verfahren nicht zwangsläufig von jeder Position aus den *kürzesten* Weg zum Ziel weist. Auch in Abbildung 7.7 ist ein Umweg versteckt. Finden Sie ihn? Die durch Pfeile markierten Wege haben aber *immer* eine endliche Länge. Eine bloß zufällige Suchbewegung hingegen kann im Prinzip jede beliebige Länge annehmen und daher auch unendlich lange dauern.

Wir müssen an dieser Stelle zudem einen Punkt betonen, der für das Verständnis des Q-Learnings von entscheidender Bedeutung ist: Aufgrund seiner Kurzsichtigkeit kann unser Eichhörnchen immer nur die eine Zelle sehen, auf der es sich gerade befindet. Trotzdem gelingt es ihm schließlich, alle Zellen im Garten mit Pfeilen zu markieren, die jederzeit den richtigen Weg weisen. Der Trick besteht – wie schon gezeigt – darin, dass das Eichhörnchen, wenn es auf einen Pfeil oder das Nussversteck gestoßen ist, jeweils einen Schritt zurückgeht und einen Pfeil in Richtung Fundstelle zeichnet. *Die Übertragung von Information findet also immer nur zwischen unmittelbar benachbarten Zellen statt.* Wie Sie später sehen werden, basiert auch das Q-Learning auf diesem Prinzip.

© Rheinwerk Verlag, Bonn 2023

7.2 Umwelt, Agent, Aktion und Belohnung

Eine Anwendung des Q-Learnings besteht aus vier wesentlichen Elementen: der veränderlichen *Umwelt*, dem *Agenten*, den möglichen *Aktionen* und den *Belohnungen*. Dies lässt sich auch auf das Eichhörnchenbeispiel übertragen. Kurz gesagt, stellt der Garten die Umwelt dar, und das Eichhörnchen ist der Agent. Als Aktionen stehen dem Eichhörnchen Bewegungen in die vier Richtungen hoch, runter, links und rechts zur Verfügung. Die Belohnungen sind die Nüsse.

Die *Umwelt* befindet sich zu jedem Zeitpunkt in einem definierten *Zustand*. Die Anzahl dieser Zustände ist *endlich*. Dem Agenten steht eine ebenfalls *endliche Anzahl an Aktionen* zur Verfügung, um mit der Umwelt zu interagieren. Aktionen können den Zustand der Umwelt verändern.

Manche Zustände sind mit einer Belohnung für den Agenten verknüpft. Die Belohnungen sind Teil der Umwelt. Wenn der Agent einen der vorher festgelegten *Endzustände* erreicht, ist eine Episode abgeschlossen.

Übertragen auf das Eichhörnchenbeispiel gilt:

- Es gibt 20 Umweltzustände, da der Garten aus 20 Zellen besteht, auf denen sich das Eichhörnchen befinden kann. Der Zustand der Umwelt ist in unserem Beispiel schlichtweg die Zelle, auf der sich das Eichhörnchen gerade aufhält.
- Entsprechend können Aktionen des Eichhörnchens den Zustand der Umwelt verändern. Das ist aber nicht immer der Fall: Wenn das Eichhörnchen sich bereits ganz links befindet, verändert die Aktion »links« den Umweltzustand nicht.
- Der mit einer Belohnung verknüpfte Zustand ist erreicht, wenn das Eichhörnchen das Nussversteck gefunden hat.
- Dieser Zustand ist zudem ein Endzustand. Wenn das Eichhörnchen ihn erreicht, ist die Episode abgeschlossen.

Beim Q-Learning geht es in der Regel nicht nur darum, im nächsten Schritt eine kurzfristige Belohnung zu ergattern. Stattdessen ist es das Ziel des Agenten, eine *Strategie* zu finden, die den langfristig zu erwartenden Gewinn maximiert – da es im Fall unseres Eichhörnchens nur ein einziges Ziel gibt, sind kurz- und langfristige Belohnungen allerdings identisch.

Durch wiederholtes Durchlaufen der Umwelt soll der Agent lernen, für jeden möglichen Umweltzustand jene Aktion zu wählen, die den langfristig erwartbaren Gewinn maximiert.

Persönliches Exemplar für Pit Noack

7.2.1 Das Verhältnis von Agent und Umwelt

Agent und Umwelt beeinflussen sich gegenseitig. Wenn der Agent eine Aktion ausführt, aktualisiert sich der Zustand der Umwelt. Dieser neue Zustand bestimmt wiederum, welche Aktion der Agent als Nächstes wählt. Dieses rekursive Wechselverhältnis ist in Abbildung 7.8 dargestellt.

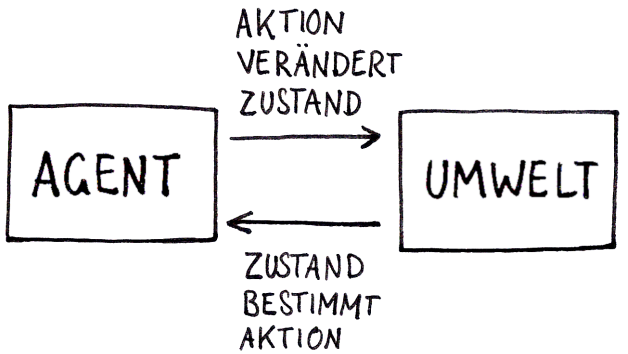


Abbildung 7.8 Agent und Umwelt wirken aufeinander ein.

Im Eichhörnchenbeispiel ist die Position des Agenten zugleich der Umweltzustand. Insofern könnten wir genauso gut sagen, dass der Agent ein Teil der Umwelt ist.

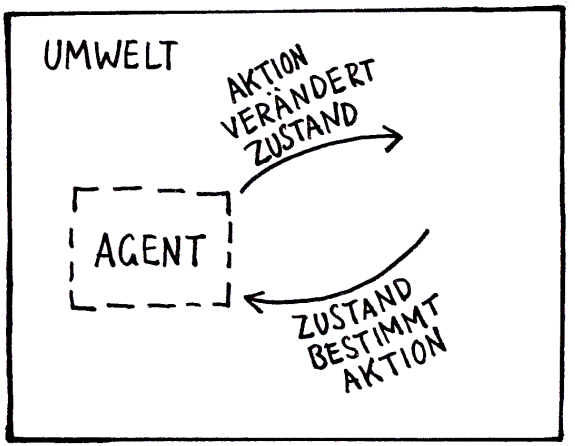


Abbildung 7.9 Wir könnten auch sagen, dass der Agent Teil der Umwelt ist.

Aus der Perspektive einer Softwareentwicklerin ist es allerdings meist sinnvoll, Agent und Umwelt getrennt voneinander zu betrachten. So lassen sich für dieselbe Umgebung verschiedene Agenten programmieren. Auch dies ließe sich auf unser Beispiel übertra-

© Rheinwerk Verlag, Bonn 2023

gen. Schließlich ist nicht nur das Eichhörnchen an den Nüssen im Garten interessiert. Manche Vogelarten lassen sich gerne hin und wieder eine Nuss schmecken und könnten das Versteck aufspüren und plündern.

7.3 Die Q-Tabelle

Unser Eichhörnchenbeispiel demonstriert eine stark vereinfachte Variante des Q-Learnings – wiederum anhand einer sehr einfachen Aufgabe. Dennoch lassen sich daran viele wesentliche Begriffe verdeutlichen – so auch der Begriff *Q-Tabelle*. Wir können die »Lösung« des Nussproblems in Form einer Tabelle darstellen, wie sie Abbildung 7.10 zeigt.

ZUSTÄNDE	AKTIONEN: hoch	runter	links	rechts
0	0	1	0	0
1	0	0	0	1
2	0	0	0	1
3	0	1	0	0
4	0	1	0	0
5	0	0	0	1
6	0	1	0	0
7	0	1	0	0
8	0	1	0	0
9	0	0	1	0
10	1	0	0	0
11	0	0	0	1
12	0	0	0	1
13	0	0	0	0
14	0	0	1	0
15	0	0	0	1
16	1	0	0	0
17	0	0	0	1
18	1	0	0	0
19	1	0	0	0

Abbildung 7.10 Jede Zeile der Tabelle steht für einen Zustand der Umwelt, jede Spalte für eine Aktion. Im Fall des Eichhörnchenbeispiels ist die bevorzugte Aktion mit einer 1 markiert.

Persönliches Exemplar für Pit Noack

In Abbildung 7.1 haben Sie bereits gesehen, dass die Zellen des Gartens nummeriert sind. Passend dazu haben wir auch die Zeilen in der Tabelle mit Nummern versehen: Wenn sich das Eichhörnchen auf Zelle 0 befindet, ist die Umwelt in Zustand 0, wenn es sich auf Zelle 1 befindet, ist sie in Zustand 1 etc. Also hat die Tabelle 20 Zeilen für 20 Umweltzustände und 4 Spalten für die 4 Aktionen *hoch*, *runter*, *links* und *rechts*. *Die Zeilen einer Q-Tabelle stehen für die Umweltzustände, die Spalten für die möglichen Aktionen.*

7.3.1 Q steht für Qualität

In der Tabelle stehen die sogenannten *Q-Werte*. Q steht für Qualität. Ein Q-Wert sagt uns, wie sinnvoll eine Aktion in einem bestimmten Zustand ist, und zwar in Hinblick auf zukünftige Belohnungen. In unserer vereinfachten Variante steht in jeder Zeile eine 1, alle anderen Felder sind mit 0 gefüllt. Die 1 steht für die Richtung des Pfeils in der entsprechenden Zelle, wie Sie durch einen Vergleich mit Abbildung 7.7 feststellen können.

Beim tatsächlichen Q-Learning speichert die Q-Tabelle auch von 0 und 1 abweichende Werte. Die Berechnung dieser Werte zeigen wir in Abschnitt 7.5, »Die Q-Tabelle befüllen«. Zunächst schauen wir uns aber das Beispielprogramm »Q-Lerner« an.

7.4 Das Beispielprogramm Q-Lerner

Dieses Programm setzt eine erweiterte Fassung des Nussproblems um. Hier wird der Agent durch ein Quadrat dargestellt, das in Abbildung 7.11 mit dem Buchstaben A markiert und im Programm blau eingefärbt ist. Der Agent startet jede Episode am linken Spielfeldrand. Das Ziel ist ein Quadrat (Z, am Bildschirm grün) am rechten Spielfeldrand. Zwischen Agent und Ziel befinden sich drei Barrieren aus Quadraten (X, am Bildschirm rot), die vom Agenten nicht passiert werden können.

Zu Beginn des Programms ist der Agent noch völlig unbedarft und springt orientierungslos herum. Irgendwann jedoch überwindet er mit diesen hilflos wirkenden Bewegungen durch bloßen Zufall die drei Barrieren und erreicht das Ziel. Die erste Episode ist damit abgeschlossen. *Genau wie im Eichhörnchenbeispiel positioniert der Agent mit jeder Episode einen weiteren Pfeil in Richtung Ziel.* Anschließend springt der Agent auf eine neue Startposition. Damit beginnt eine weitere Episode.

© Rheinwerk Verlag, Bonn 2023

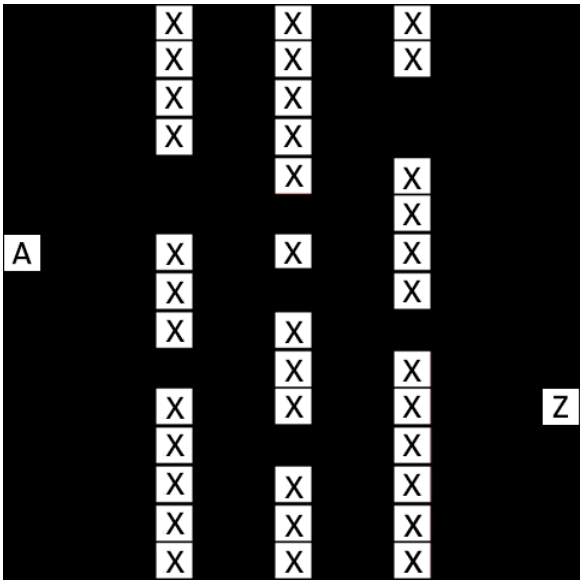


Abbildung 7.11 Das Spielfeld des Beispielprogramms Q-Lerner zu Beginn einer Episode und bevor irgendetwas gelernt wurde

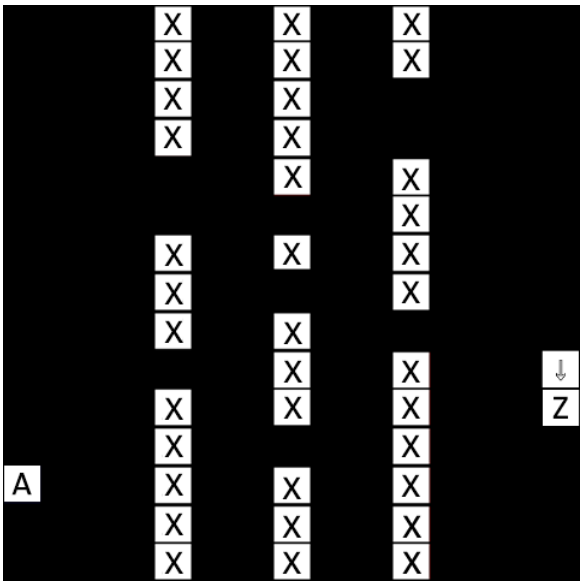


Abbildung 7.12 Die erste Episode ist mit dem Auffinden des Ziels und der Positionierung eines Pfeils in Richtung Ziel abgeschlossen; der Agent springt zurück auf eine neue Startposition.

Persönliches Exemplar für Pit Noack

Abbildung 7.13 und Abbildung 7.14 zeigen die Zustände jeweils unmittelbar nach Abschluss der zweiten und dritten Episode.

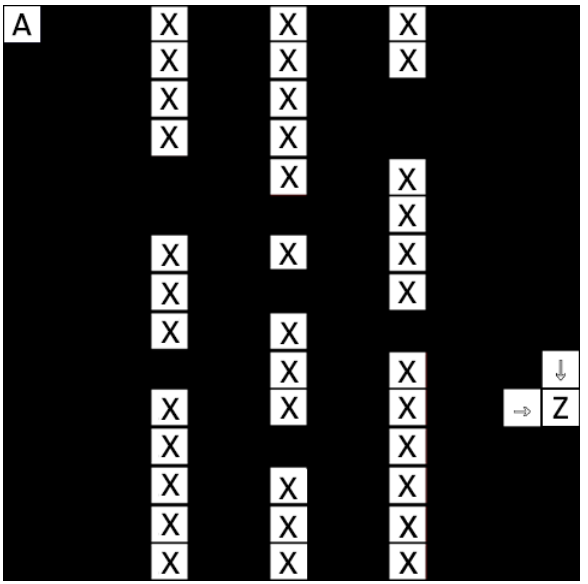


Abbildung 7.13 Mit dem Abschluss einer jeden Episode ...

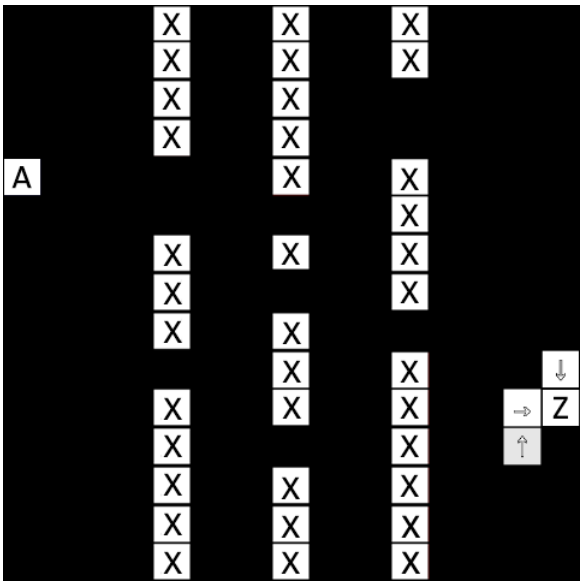


Abbildung 7.14 ... zeichnet der Q-Lerner einen neuen Pfeil.

© Rheinwerk Verlag, Bonn 2023

In Abbildung 7.15 sehen Sie die Veränderung des Spielfelds im weiteren Verlauf des Programms. Zuletzt sind alle Zellen mit Pfeilen versehen, die den Weg Richtung Ziel weisen.

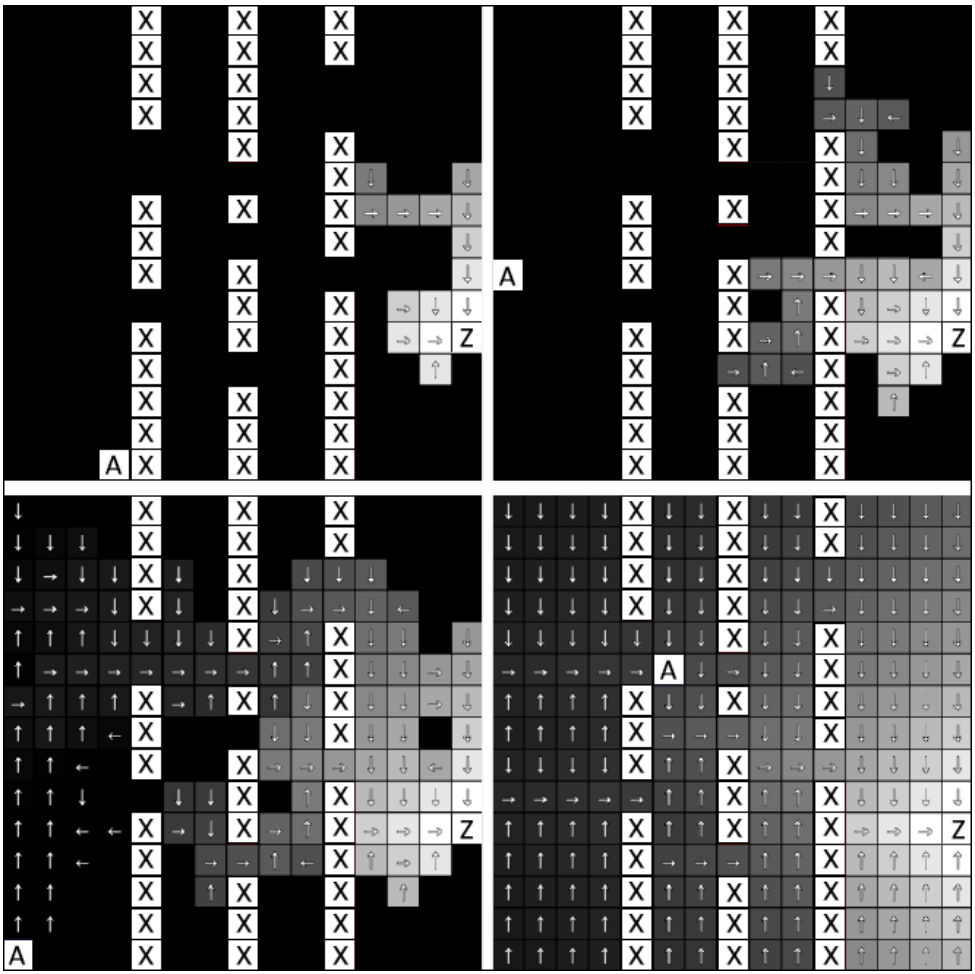


Abbildung 7.15 Je länger der Q-Lerner lernt, umso mehr Zellen sind mit Pfeilen markiert.

Sie sehen in der Abbildung, dass die Zellen des Spielfelds mit unterschiedlich hellen Graustufen eingefärbt sind. Die Helligkeit einer Zelle steht für deren Q-Wert: Je näher eine Zelle dem Ziel ist, umso heller leuchtet sie. Die Pfeile zeigen jeweils auf diejenige benachbarte Zelle, die den höchsten Q-Wert besitzt.

Die unterschiedlichen Helligkeitsstufen weisen auf einen fundamentalen Unterschied zu der Q-Tabelle hin, die wir für das Nussproblem erstellt haben: Dort nämlich konnten die

Persönliches Exemplar für Pit Noack

Einträge nur zwei unterschiedliche Werte annehmen, 0 oder 1 – wobei die jeweils bevorzugte Aktion mit 1 gekennzeichnet ist.

Unser Q-Lerner erstellt im Hintergrund ebenfalls eine Q-Tabelle. Genau wie in Abbildung 7.10 gezeigt, steht dort jede Zeile für einen Umweltzustand und jede Spalte für eine Aktion. Im Unterschied zum Nussproblem speichert die Tabelle für jedes Zustands-Aktions-Paar eine Dezimalzahl zwischen 0 und 100. Die Q-Werte können also jeden beliebigen Wert dazwischen annehmen, zum Beispiel 13,45 oder 72,9. Der maximale Q-Wert 100 kommt daher, dass wir die Belohnung für den Zielzustand auf 100 gesetzt haben.

Genau wie bei der Lösung des Nussproblems wird auch beim Q-Lerner die Information nur zwischen benachbarten Zellen übertragen. Benachbarte Zellen entsprechen Zuständen, die nur eine einzige Aktion voneinander entfernt sind. Wie diese Informationsübertragung abläuft, wie also der Q-Lerner die Q-Tabelle befüllt, zeigen wir im folgenden Abschnitt.

Vorher präsentieren wir noch einige nützliche Funktionen des Q-Learners: Wenn Sie auf irgendeine der Zellen klicken, können Sie die Zeile der Q-Tabelle auslesen, die dieser Zelle zugeordnet ist. Im Textfeld unten erscheint ein kurzer Bericht, der neben der Zeilen- und der Spaltennummer die der Zelle entsprechende Zustandsnummer und die Q-Werte für die vier möglichen Aktionen darstellt. In der Abbildung ist der Wert für die Aktion *runter* am höchsten. Entsprechend wird auch der Pfeil in dieser Zelle nach unten zeigen.

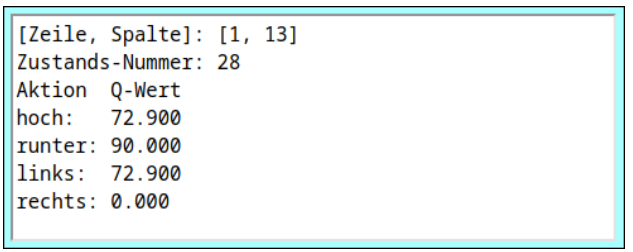


Abbildung 7.16 Das Textfeld zeigt einen Bericht über eine angeklickte Zelle.

Beim Ausprobieren des Q-Lerners werden Sie feststellen, dass das Programm sehr viele Lernschritte und Episoden braucht, um das Spielfeld mit Pfeilen zu füllen. Daher haben wir den Button 5000 Schritte eingebaut, mit dem Sie den Lernprozess erheblich beschleunigen können.

© Rheinwerk Verlag, Bonn 2023

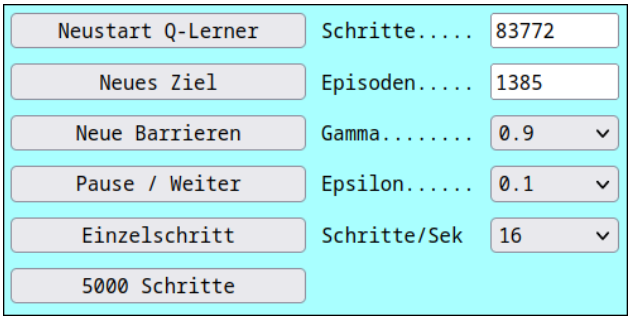


Abbildung 7.17 Die grafischen Bedienelemente des Q-Lerners

Die anderen Buttons in der linken Spalte sind selbsterklärend. In der rechten Spalte wird jeweils die Anzahl der Lernschritte und der Episoden angezeigt. Mit dem Drop-down-Menü SCHRITTE/SEK bestimmen Sie das Tempo, mit dem der Agent über das Spielfeld flitzt. Was es mit GAMMA und mit EPSILON auf sich hat, erklären wir in Abschnitt 7.7 und Abschnitt 7.8.

7.5 Die Q-Tabelle befüllen

Nun nähern wir uns der alles entscheidenden Frage: Wie füllt der Q-Lerner in unserem Beispielprogramm die Q-Tabelle?

Im Eichhörnchenbeispiel wurde jeweils mit dem Abschluss einer Episode ein Pfeil gezeichnet beziehungsweise eine 1 in die entsprechende Zelle der Q-Tabelle eingetragen. Ein einmal gezeichneter Pfeil beziehungsweise eine eingetragene 1 blieb unveränderlich.

Das Vorgehen des Q-Lerners im Beispielprogramm unterscheidet sich in diesem sehr wesentlichen Punkt: Mit *jeder* Aktion des Agenten wird das entsprechende Feld in der Q-Tabelle aktualisiert.

Abbildung 7.18 zeigt die Formel, mit der unser Q-Lerner einen Tabelleneintrag aktualisiert. Die Formel ist weit weniger kompliziert, als sie auf den ersten Blick erscheinen mag. Schauen wir uns die Bestandteile der Formel der Reihe nach an:

- (z_t, a_t) bezeichnet das aktuelle *Zustands-Aktions-Paar*.
- $Q_{neu}(z_t, a_t)$ ist der neue Wert der Zelle der Q-Tabelle, die aktualisiert werden soll.

Persönliches Exemplar für Pit Noack

Zur Berechnung dieses neuen Werts benötigen wir drei Komponenten:

- b steht für die Belohnung, die der Agent für das aktuelle Zustands-Aktions-Paar erhält.
- γ (*Gamma*) steht für die *Verminderungsrate*. Sie regelt kurz gesagt, in welchem Maße künftige Belohnungen Einfluss auf die Qualität des aktuellen Zustand-Aktions-Paars haben – dazu mehr in Abschnitt 7.7.
- z_{t+1} ist der Zustand, der durch das aktuelle Zustand-Aktions-Paar erreicht wird. Wir nennen ihn *Folgezustand*. $Q_{\max}(z_{t+1})$ ist dann der höchste Wert in der Zeile der Q-Tabelle, die zum diesem Folgezustand gehört.

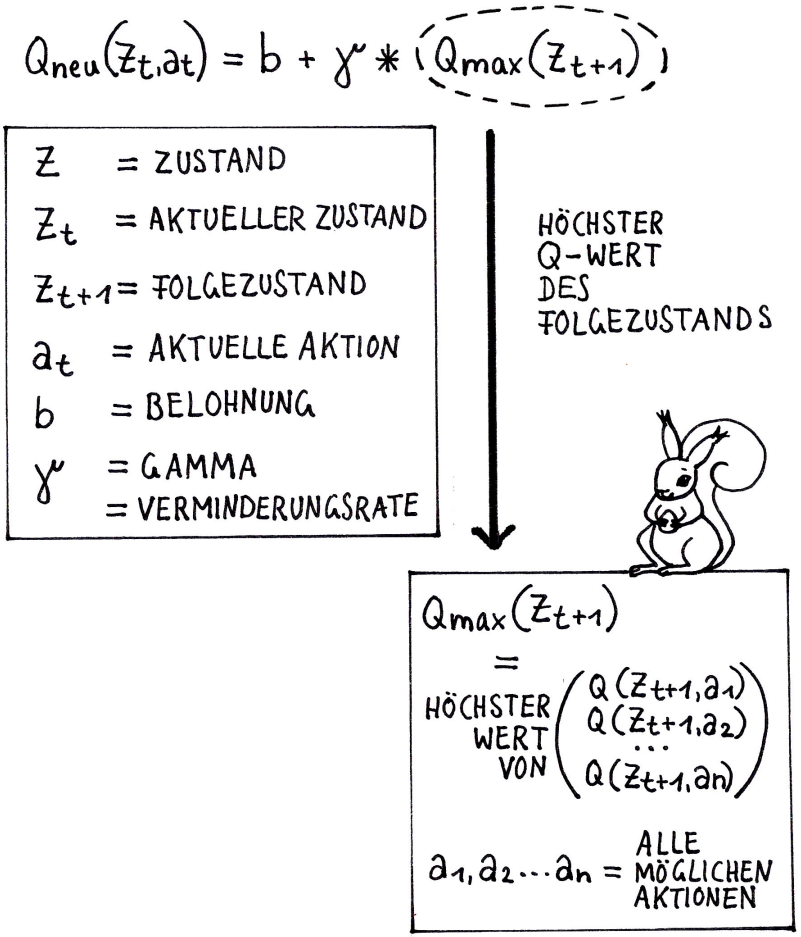


Abbildung 7.18 Die Formel für die Aktualisierung einer Zelle der Q-Tabelle

© Rheinwerk Verlag, Bonn 2023

Wir erhalten den neuen Wert der Zelle, indem wir den höchstmöglichen Q-Wert des Folgezustands mit γ multiplizieren und die Belohnung für den aktuellen Zustand addieren.

Der Q-Wert eines Zustands-Aktions-Paars hängt also sowohl von der direkten Belohnung als auch von den zukünftig zu erwartenden Belohnungen ab. In jedem Lernschritt wird der Q-Wert des entsprechenden Felds in der Tabelle erneut ermittelt. Die Q-Werte sind also nicht konstant, sondern verändern sich im Laufe des Lernens. In Fall unseres Beispielprogramms stabilisieren sich die Werte, wenn der Agent lang genug lernt und eine optimale Strategie gefunden hat.

7.5.1 Warum funktioniert das?

Damit ist die mathematische Seite der Aktualisierung der Q-Tabelle geklärt. Aber *warum* erreicht der Q-Lerner mittels dieser Berechnung von Q-Werten das Ziel, für jeden Zustand die vielversprechendste Aktion zu finden – vorausgesetzt, wir lassen ihn lang genug probieren?

Abbildung 7.15 kann ein intuitives Verständnis vermitteln: Sie sehen deutlich einen Farbverlauf bei den Einfärbungen der einzelnen Zellen: Je weiter diese vom Ziel entfernt sind, umso dunkler ist die Zelle gefärbt.

Stellen Sie sich vor, dass der Zielzustand ein in Auflösung befindliches Salzkristall in einem sehr großen Wasserbehälter ist, in dem keine Strömung herrscht. Je näher wir dem Kristall kommen, desto höher wird die Salzkonzentration im Wasser sein. Ein solcher Auflösungs- und Verdünnungsprozess wird auch *Diffusion* genannt. Und ganz ähnlich diffundiert die Qualität eines mit einer Belohnung verknüpften Zustands durch den *Zustandsraum* unserer Umwelt.

Im Beispiel mit dem Salzkristall sorgt die selbsttätige Molekularbewegung für die Verteilung der Salzmoleküle im Wasserbecken. Beim Q-Learning mittels Q-Tabelle ist es der Agent, der durch fortwährendes Aktualisieren der Q-Tabelle die Qualität des Zielzustands im Zustandsraum verteilt. Und zwar – wir betonen diesen wichtigen Punkt hier gerne noch einmal – immer zwischen Zuständen, die nur eine Aktion voneinander entfernt sind.

Das Beispiel mit dem Salzkristall ist noch in einer anderen Hinsicht passend. Wenn Sie zum Salzkristall gelangen wollen, werden Sie sich immer in die Richtung bewegen, in der die Salzkonzentration im Wasser zunimmt. Genau das tut der Q-Lerner, wenn er den Pfeilen folgt.

Persönliches Exemplar für Pit Noack

7.6 Der Code unter der Lupe

Im Code des Beispielprogramms sind Objekte der Klassen `Umwelt` und `QLerner` entscheidend. Sie finden die entsprechenden Definitionen in den Dateien *code/umwelt.js* und *code/qlerner.js*.

7.6.1 Die Umwelt

Die in Kapitel 5, »Spiele für eine Person lösen«, und Kapitel 6, »Spiele für zwei Personen gewinnen«, vorgestellten KIs können verschiedene Spiele spielen, sofern diese eine passende Schnittstelle bereitstellen. Sehr ähnlich läuft es bei unserem Q-Lerner: Er kann verschiedene Aufgaben bewältigen, also mit verschiedenen Umwelten interagieren, vorausgesetzt, die Schnittstelle passt.

Tabelle 7.1 zeigt die Eigenschaften und Funktionen von Objekten der Klasse `Umwelt`, die diese Schnittstelle bilden.

Name	Aufgabe
`nZustaende`	Anzahl der möglichen verschiedenen Zustände der Umwelt.
`nAktionen`	Anzahl der Aktionen, die ein Agent wählen kann.
`aktion(aktionsNr)`	Führt die Aktion mit der entsprechenden `aktionsNr` aus und modifiziert die Umwelt. Rückgabewert ist die modifizierte Umwelt.
`zustandsNr()`	Liefert die Nummer des aktuellen Zustands.
`belohnung()`	Liefert die Belohnung für den aktuellen Zustand.

Tabelle 7.1 Funktionen von Objekten der Klasse Umwelt, die die Schnittstelle für den Q-Lerner bilden

Listing 7.1 zeigt den Code der Funktion `umwelt.belohnung()`. Wenn die Position des Agenten identisch ist mit der Position der Belohnung, gibt die Funktion `100` zurück, sonst `0`.

```
belohnung() {
  if (this.posAgent.zeile == this.posZiel.zeile &&
    this.posAgent.spalte == this.posZiel.spalte) {
    return 100;
```


© Rheinwerk Verlag, Bonn 2023

```
  }
  return 0;
}
```

Listing 7.1 Die Funktion umwelt.belohnung()

7.6.2 Der Q-Lerner

Tabelle 7.2 und Tabelle 7.3 zeigen wesentliche Eigenschaften und Funktionen von Objekten der Klasse `QLerner`.

Name	Inhalt
`gamma`	Legt fest, wie weit die Qualität künftiger Zustände auf die Qualität des aktuellen Zustands einwirkt (siehe Abschnitt 7.7); Wertebereich zwischen 0 und 1.
`epsilon`	Bestimmt die Experimentierfreudigkeit des Q-Lerners (siehe Abschnitt 7.8); Wertebereich zwischen 0 und 1.
`nAktionen`	Die Anzahl der möglichen Aktionen.
`qTabelle`	Die Q-Tabelle, die der Q-Lerner befüllt.

Tabelle 7.2 Wesentliche Eigenschaften ...

Name	Aufgabe
`schritt(umwelt)`	Vollzieht einen vollständigen Schritt, der aus der Wahl einer Aktion, deren Ausführung und der Aktualisierung der Q-Tabelle besteht. Ruft `waehleAktion()`, `umwelt.aktion()` und `aktualisiereQ()` auf.
`waehleAktion(zustandsNr)`	Wählt eine Aktion abhängig von `zustandsNr`.
`aktualisiereQ(zustandsNr, aktionsNr, folgezustandsNr, belohnung)`	Aktualisiert die Q-Tabelle in der Zeile `zustandsNr` und der Spalte `aktionsNr`.

Tabelle 7.3 ... und Funktionen von Objekten der Klasse QLerner

Persönliches Exemplar für Pit Noack

Jedes Mal, wenn der Agent »am Zug« ist, wird die Funktion `schritt()` aufgerufen (siehe Listing 7.2). In einem Schritt werden drei Dinge erledigt:

1. Wahl einer Aktion abhängig vom Zustand der Umwelt
   - `umwelt.zustandsNr()` liefert die Nummer des aktuellen Umweltzustands.
   - `waehleAktion()` wählt dann die Aktion abhängig von der `zustandsNr` aus.
2. Anwendung der Aktion
   - `umwelt.aktion()` wendet die gewählte Aktion auf die Umwelt an.
3. Modifikation der Umwelt
   - Die modifizierte Umwelt wird unter `folgezustand` gespeichert.
   - `folgezustand.zustandsNr()` liefert die Nummer des Folgezustands.
   - `folgezustand.belohnung()` liefert die mit der letzten Aktion erlangte Belohnung.
   - `aktualisiereQ()` aktualisiert die Q-Tabelle.

```
schritt(umwelt) {
  const zustandsNr = umwelt.zustandsNr();
  const aktionsNr = this.waehleAktion(zustandsNr);
  const folgezustand = umwelt.aktion(aktionsNr);
  const folgezustandsNr = folgezustand.zustandsNr();
  const belohnung = folgezustand.belohnung();
  this.aktualisiereQ(zustandsNr, aktionsNr, folgezustandsNr, belohnung);
  this.nSchritte += 1;
  return folgezustand;
}
```

Listing 7.2 Die Funktion qlerner.schritt()

Die Funktion `schritt()` ruft die Funktionen `waehleSchritt()` und `aktualisiereQ()` auf. Bevor wir uns den Code dieser beiden Funktionen anschauen, müssen wir noch klären, was es mit Gamma und Epsilon auf sich hat.

7.7 Gamma bestimmt die Weitsicht

Gamma wird auch *Verzögerungsrate* genannt. Sie bestimmt, wie weit in die Zukunft mögliche Belohnungen beachtet werden.

Beträgt Gamma 0, interessiert sich der Agent nur für die unmittelbar mit einer Aktion verbundenen Belohnungen, nicht aber für Belohnungen, die nach drei, fünf oder sechsundzwanzig Schritten winken. Somit lernt der Agent nur sehr wenig. Lediglich die Q-

© Rheinwerk Verlag, Bonn 2023

Werte von Zuständen, die nur einen einzigen Schritt von einem Belohnungszustand entfernt sind, können überhaupt verändert werden. Das Ergebnis sehen Sie in Abbildung 7.20 links.

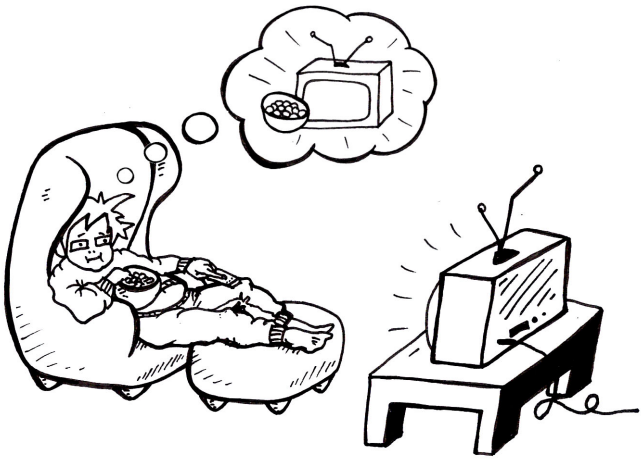


Abbildung 7.19 Bei einem niedrigen Wert für Gamma interessiert sich der Q-Lerner nur für nahe liegende Belohnungen.

Beträgt Gamma 1, interessiert sich ein Agent für alle möglichen Langzeitfolgen. Die Anzahl an Aktionen, die für das Erreichen des Ziels nötig ist, spielt keine Rolle. Es macht überhaupt keinen Unterschied, wohin sich der Agent bewegt, weil jeder zufällig gewählte Weg *irgendwann* zum Ziel führen wird.

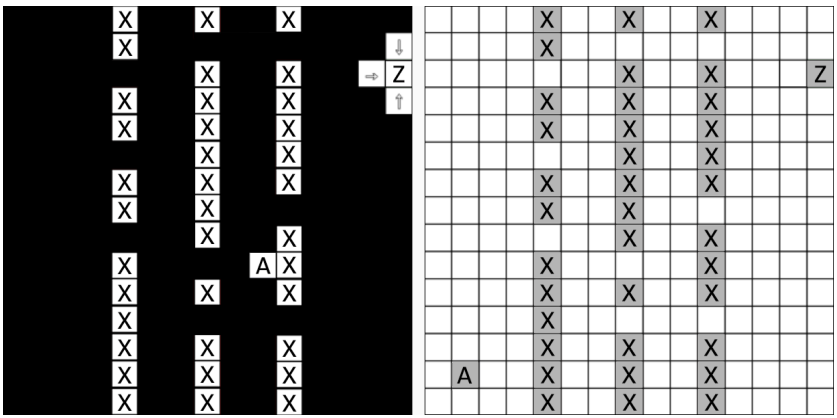


Abbildung 7.20 Bei Gamma = 0 erhalten nur die dem Belohnungszustand benachbarten Zustände einen Q-Wert. Bei Gamma = 1 erhalten im Extremfall alle Zustände einen maximalen Q-Wert. Beides bietet keine Orientierung.

Persönliches Exemplar für Pit Noack

Abbildung 7.20 rechts zeigt, was passiert, wenn Sie im Q-Lerner Gamma auf 1 setzen und den Agenten lang genug lernen lassen: Alle Zellen haben den Q-Wert 100 – was der Höhe der Belohnung des Zielzustands entspricht. Richtungspfeile sind nicht mehr zu sehen, da diese nur angezeigt werden, wenn es eine bevorzugte Richtung gibt.

Beide Szenarien sind offensichtlich nicht ideal. In der Praxis wählen wir daher Gamma-Werte, die weder 0 noch 1 sind. Bewährt haben sich Werte zwischen 0,9 und 0,99.



Abbildung 7.21 Ein Gamma-Wert um 0,9 erlaubt dem Q-Lerner, entferntere Ziele mit einem angemessenen Aufwand anzustreben.

7.8 Epsilon: Erforschung oder Anwendung

Der Wert *Epsilon* bestimmt, ob sich der Agent auf zufälliges Erforschen der Umgebung oder auf das Anwenden der besten Aktion konzentriert. Je höher der Epsilon-Wert, umso experimentierfreudiger ist der Q-Lerner.

© Rheinwerk Verlag, Bonn 2023

Ist Epsilon 1, führt der Agent *nur* zufällige Aktionen aus. Er füllt seine Q-Tabelle zwar mit sinnvollen Werten, wendet das so gewonnene Wissen aber nie an. Ein solcher Agent ist durch ineffizientes Verhalten charakterisiert.

Übertragen auf den Q-Lerner bedeutet das: Nach hinreichend langem Training sind zwar alle Zellen mit Pfeilen gefüllt, die in die jeweils vielversprechendste Richtung weisen, allerdings hält sich der Agent nicht an diese Wegweiser, sondern springt genauso ziellos umher wie ein Agent, der noch gar nichts gelernt hat.

Ist Epsilon hingegen 0, findet keine umfassende Erforschung der Umwelt statt. Der Agent entscheidet sich dann immer für die von der Q-Tabelle empfohlene Aktion – einmal gelernte Umwege bleiben bestehen. Auch dies bereitet Schwierigkeiten. Zwar werden die ersten Züge auch in diesem Fall zufällig ausgeführt, da alle Werte in der Q-Tabelle 0 sind, aber sobald die Q-Tabelle teilweise befüllt ist, wird der Agent sich vorwiegend auf festgefahrenen Wegen bewegen. Ein solcher Agent ist in der Regel nicht in der Lage, die optimale Lösung zu finden, und wird für die meisten Zustände keinen Q-Wert ermitteln.

Abbildung 7.22 zeigt ein mögliches Ergebnis eines Q-Lerners, bei dem Epsilon auf 1 eingestellt ist. Sie können zahlreiche Pfeile finden, die auf einen Umweg verweisen. Viele Zellen (= Umweltzustände) wurden nie erreicht und sind entsprechend auch nicht mit Pfeilen markiert.

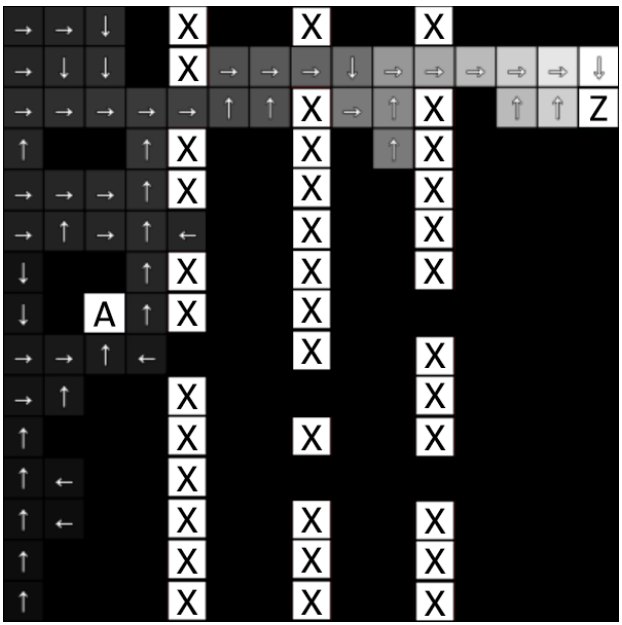


Abbildung 7.22 Bei einem niedrigen Epsilon-Wert findet der Q-Lerner keine optimale Lösung; viele Zustände verbleiben ohne jeden Hinweis auf eine bevorzugte Richtung.

Persönliches Exemplar für Pit Noack

Für Epsilon werden in der Praxis entsprechend Werte verwendet, die zwischen 0 und 1 liegen. Ein Wert um 0,1 hat sich bewährt. Ein Epsilon von 0,1 sagt aus, dass der Agent in 10 % der Fälle mit einer zufällig gewählten Aktion die Umgebung erforscht und in 90 % der Fälle die von der Q-Tabelle empfohlene Aktion ausführt.

Es ergibt oft Sinn, dass ein Agent am Anfang gründlich die Umgebung erforscht und im Verlauf des Trainings immer mehr dazu übergeht, das Gelernte anzuwenden. Dies kann man durch eine kontinuierliche Verminderung von Epsilon erreichen: Der Agent beginnt mit einem hohen Epsilon-Wert. Mit jeder Episode wird der Wert um einen geringen Betrag verringert, bis er bei einem niedrigen Wert – zum Beispiel 0,1 – konstant bleibt.



Abbildung 7.23 Bei einem hohen Wert für Epsilon ist der Agent experimentierfreudiger.

7.9 Ein zweiter Blick auf den Code

Jetzt haben wir geklärt, wofür die Werte Gamma und Epsilon stehen, und können uns anschauen, wie die Funktionen `waehleAktion()` und `aktualisiereQ()` ihr Werk verrichten.

© Rheinwerk Verlag, Bonn 2023

Listing 7.3 zeigt die Funktion `waehleAktion()`. Ihr Rückgabewert kann vom Zustand der Umwelt abhängen oder per Zufall bestimmt werden. Die Funktion trifft die Entscheidung wie folgt:

- `this.qTabelle[zustandsNr]` liefert die Zeile der Q-Tabelle, die zum aktuellen Zustand der Umwelt gehört.
- Wenn alle Werte in dieser Zeile gleich sind, bietet die Q-Tabelle für den entsprechenden Zustand keinerlei Orientierung – das prüft unsere Hilfsfunktion `alleGleich()`. In diesem Fall liefert die Funktion mit `int(random(this.nAktionen))` eine zufällig ausgewählte Aktionsnummer.
- Andernfalls geht es mit der zweiten `if`-Abfrage weiter: `random()` liefert eine Zufallszahl zwischen 0 und 1. Ist diese kleiner als `epsilon`, ist der Rückgabewert der Funktion ebenfalls eine zufällig ausgewählte Aktionsnummer. Bei einem Epsilon-Wert von 0,1 ergibt `waehleAktion()` also in 10 % aller Fälle eine zufällige Aktion.
- Die letzte Zeile des Funktionskörpers kommt nur dann zum Zuge, wenn die beiden `if`-Abfragen `false` waren. In diesem Fall kann und soll also die Q-Tabelle den Weg weisen. Unsere Hilfsfunktion `maxPos()` liefert dann die Position des höchsten Werts jener Zeile der Q-Tabelle, die zum aktuellen Umweltzustand gehört. Das ist die Nummer der bevorzugten Aktion.

```
waehleAktion(zustandsNr) {

  const qTabellenZeile = this.qTabelle[zustandsNr];

  if (alleGleich(qTabellenZeile)) {
    return int(random(this.nAktionen));
  }

  if (random() < this.epsilon) {
    return int(random(this.nAktionen));
  }

  return maxPos(qTabellenZeile);
}
```

Listing 7.3 Die Funktion qlerner.waehleAktion()

Die Funktion `aktualisiereQ()` vollzieht schließlich die Formel, die wir in Abschnitt 7.5, »Die Q-Tabelle befüllen«, vorgestellt haben, und speichert das Ergebnis im entsprechenden Feld der Q-Tabelle.

```
aktualisiereQ(zustandsNr, aktionsNr, folgezustandsNr, belohnung) {
  const qTabellenZeileFolgezustand = this.qTabelle[folgezustandsNr];
  const maxQ = max(qTabellenZeileFolgezustand);
  const qNeu = belohnung + this.gamma * maxQ;
  this.qTabelle[zustandsNr][aktionsNr] = qNeu;
}
```

Listing 7.4 Die Funktion qlernen.aktualisiereQ()

7.10 Alpha

Die Umwelt in unserem Beispielprogramm ist vollständig determiniert. Das heißt, dass der Folgezustand einer Aktion einzig und allein vom vorherigen Umweltzustand und der gewählten Aktion des Agenten abhängt. Eine Umwelt, bei der das nicht der Fall ist, lässt sich leicht vorstellen: Was wäre etwa, wenn das Ziel genauso wie der Agent in der Lage wäre, gelegentlich die Position zu wechseln? Oder wenn andere Agenten die Umwelt bewohnen und die Interessen unseres Agenten zu durchkreuzen suchen? Dann würden die Folgezustände nicht mehr allein von den Aktionen des Agenten abhängen.

Auch für solche Umwelten können wir per Q-Learning mittels Q-Tabelle sinnvolle Strategien finden. Allerdings ist dafür eine Erweiterung unserer Aktualisierungsfunktion erforderlich, die wir hier kurz vorstellen.

Abbildung 7.24 zeigt die erweiterte Gleichung. Der Unterschied zur vorher verwendeten Gleichung: Zur Aktualisierung eines Felds wird dessen vorheriger Wert nicht mehr einfach überschrieben. Vielmehr setzt sich dieser aus dem neu berechneten Wert *und* dem vorherigen Wert zusammen. Die Lernrate Alpha steuert das Verhältnis:

- Ist die Lernrate gleich 1, zählt allein der neu berechnete Wert. Dann liefert die Gleichung dasselbe Ergebnis wie die Variante für determinierte Umgebungen aus Abschnitt 7.5.
- Bei einer Lernrate von 0,5 bestimmen der vorherige und der neu berechnete Wert zu gleichen Teilen den neuen Q-Wert.
- Bei einer Lernrate von 0 würde sich nichts ändern. Es wird also lediglich der vorherige Wert übernommen. Das ergibt offensichtlich keinen Sinn, da überhaupt nichts gelernt werden kann.

© Rheinwerk Verlag, Bonn 2023

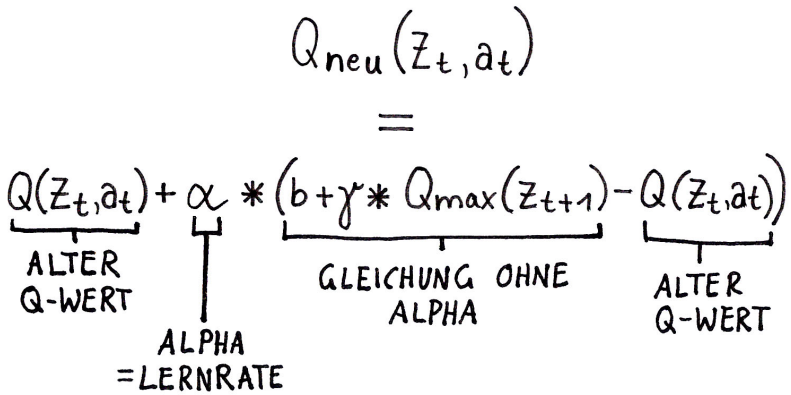


Abbildung 7.24 Die erweiterte Gleichung für Umwelten mit Zufallselementen

Sie finden im Code des Q-Lerners eine entsprechende Erweiterung der Aktualisierungsfunktion, die im aktuellen Beispielprogramm aus Mangel an Notwendigkeit allerdings keine Anwendung findet.

```
aktualisiereQAlpha(zustandsNr, aktionsNr, folgezustandsNr, belohnung) {
  const qAlt = this.qTabelle[zustandsNr][aktionsNr];
  const qTabellenZeile = this.qTabelle[folgezustandsNr];
  const maxQ = max(qTabellenZeile);
  const qNeu = qAlt + this.alpha * (belohnung + this.gamma * maxQ - qAlt);
  this.qTabelle[zustandsNr][aktionsNr] = qNeu;
}
```

Listing 7.5 Die Funktion qlerner.aktualisiereQAlpha()

7.11 Was wir weggelassen haben

In diesem Kapitel ging es uns erneut vor allem darum, grundlegende Ideen zu vermitteln. Deshalb haben wir auch in diesem Kapitel viele technische Details, Spezialfälle und Varianten unerwähnt gelassen.

Allerdings ist das Thema Q-Learning mittels Q-Tabellen ein besonders weites Feld, und daher mussten wir besonders viel auslassen und vereinfachen. Einige wichtige Punkte wollen wir an dieser Stelle zumindest kurz erwähnen:

Persönliches Exemplar für Pit Noack

7.11.1 Komplexere Umwelten

Unser Programmbeispiel zeigt die Lösung einer *äußerst* einfachen Aufgabe. Tatsächlich ist das Q-Learning mittels Q-Tabelle nicht einmal die einfachste Art, diese Aufgabe zu lösen. Mit einer schlichten Breitensuche würden wir schneller zum Ziel kommen. Wir haben uns trotzdem für eine derart simple Aufgabe entschieden, weil wir so den Zustandsraum grafisch anschaulich abbilden können und ein intuitives Verständnis des Algorithmus ermöglichen: Jede Zelle auf der Fläche ist einem Umweltzustand zugeordnet. Bei nur geringfügig komplexeren Aufgaben wäre dies nicht mehr möglich.

7.11.2 Kosten für Aktionen

Bei praktischen Anwendungen des Q-Learnings ist es oft sinnvoll, für jede ausgeführte Aktion einen Strafpunkt (also eine negative Belohnung) zu geben. Das kann die Suche nach optimalen Lösungen beschleunigen. In unserem Beispielprogramm würden wir dafür in der Funktion `umwelt.belohnung()` in Listing 7.1 lediglich den Standardrückgabewert `0` durch `-1` ersetzen. Das hat einen interessanten Effekt in der Darstellung – vielleicht mögen Sie das probieren?

7.11.3 Belohnungen mit Zustands-Aktions-Paaren verknüpfen

In unserem Beispielprogramm ist die Belohnung *direkt* mit einem Zustand verknüpft. In der Praxis sind jedoch Belohnungen meist Zustands-Aktions-Paaren zugeordnet. Dann ist die Belohnung also nicht allein vom aktuellen Zustand abhängig, sondern auch davon, *von wo* und *durch welche Aktion* der Agent zu diesem Zustand gekommen ist. Auf diese Weise ließe sich zum Beispiel eine Drehtür modellieren, die sich nur von einer Seite her passieren lässt und deren erfolgreiche Durchquerung mit einer Belohnung verknüpft ist.

Die Verknüpfung von Belohnungen mit Zustands-Aktions-Paaren wird in der Praxis häufig ebenfalls in einer Tabelle gespeichert. Das Format einer solchen Tabelle ist identisch mit jenem der Q-Tabelle.

7.11.4 Mehrere Belohnungszustände

In unserem Beispiel haben wir nur einen einzigen Zustand mit einer Belohnung verknüpft. Es kann auch mehrere Zustände dieser Art geben, und diese können mit unterschiedlich hohen Belohnungen verknüpft sein. Strafen, also Belohnungen mit umgekehrten Vorzeichen, sind ebenfalls möglich. In Kombination mit den eben erwähnten Aktionskosten können sich damit deutlich kompliziertere Strategien ergeben.

© Rheinwerk Verlag, Bonn 2023



Hintergrund: Maschinelles Lernen

Ein Computer weiß *nichts* über die Welt, in der wir leben. Ihm ist nicht bekannt, dass Bäcker Brötchen backen, dass es in Autowerkstätten nach Öl riecht und dass Elefanten in Porzellanläden nichts verloren haben. Er weiß nicht, dass alle Menschen sterben müssen, dass auf Rockkonzerten laute Musik gespielt wird und dass manche Menschen ihren Kaffee mit Milch trinken.

Eine klassische Strömung der KI-Forschung versucht, Computern solch uns Menschen selbstverständliches Alltagswissen über manuelle, stark formalisierte Eingaben beizubringen – mit dem Ziel, dieses Wissen zur Weiterverarbeitung mittels logischer Verknüpfungen bereitzustellen. Dieses Ansinnen verfolgt zum Beispiel das 1984 gestartete Projekt Cyc.

Aktuell ist in der KI-Welt ein anderer Ansatz vorherrschend, das sogenannte *maschinelle Lernen*. Hier soll uns die Mühe der manuellen Eingabe erspart bleiben. Der Computer lernt allein anhand von Beispielen oder durch Versuch und Irrtum. Dieses Vorgehen hat sich als sehr erfolgreich erwiesen. Das in diesem Kapitel vorgestellte Q-Learning ist eine Variante des maschinellen Lernens. Auch andere in diesem Buch vorgestellte Verfahren können zum maschinellen Lernen gezählt werden.

Wir finden die Unterscheidung zwischen manueller Eingabe von Wissen und Regeln einerseits und dem selbstständigen Lernen des Programms andererseits höchst bedeutsam. Allerdings halten wir den Begriff des maschinellen Lernens für nicht so trennscharf, wie er auf den ersten Blick erscheinen mag. So müssen die Daten, anhand derer gelernt werden soll, oft mit sehr großem Aufwand manuell vorbereitet werden: Damit der Computer zum Beispiel lernen kann, Katzen- und Hundebilder voneinander zu unterscheiden, müssen zuvor Tausende von Bildern entsprechend markiert worden sein. Zu dem Thema erfahren Sie mehr in Abschnitt 11.1, »Das überwachte Lernen«.

Daher sprechen wir in diesem Buch nicht von maschinellem Lernen, auch wenn der Begriff sehr verbreitet ist und in vielen Zusammenhängen geradezu als Synonym für künstliche Intelligenz verwendet wird.

7.12 Ideen zum Weitermachen

In Abschnitt 7.10, »Alpha«, haben wir eine erweiterte Formel zur Aktualisierung von Q vorgestellt, die dem Q-Lerner auch die Entwicklung von Lösungsstrategien in Umwelten mit Zufallselementen erlaubt. Ein interessantes Codeexperiment wäre die Erweiterung der Umwelt um ein Zufallselement und die Anwendung der erweiterten Funktion na-

Persönliches Exemplar für Pit Noack

mens `aktualisiereQAlpha()`. Zum Beispiel könnte das Ziel zufallsgesteuert am rechten Spielfeldrand hin- und herwandern.

7.12.1 OpenAI Gym

Wenn Sie das spannende Thema des verstärkenden Lernens vertiefen und sich gar in der Programmierung eigener Agenten üben wollen, empfehlen wir einen Blick auf die Python-Bibliothek *OpenAI Gym*. Dort finden Sie viele vollständig programmierte Umwelten. Beliebt sind etwa Atari-Spieleklassiker wie Space Invaders, Breakout oder Video Pinball, aber auch knifflige Physiksimulationen: In Mountain Car muss der Agent mit einem leistungsschwachen Fahrzeug einen Berg überwinden, beim Lunar Lander geht es darum, eine Mondfähre sicher auf unebenem Gebiet zu landen.

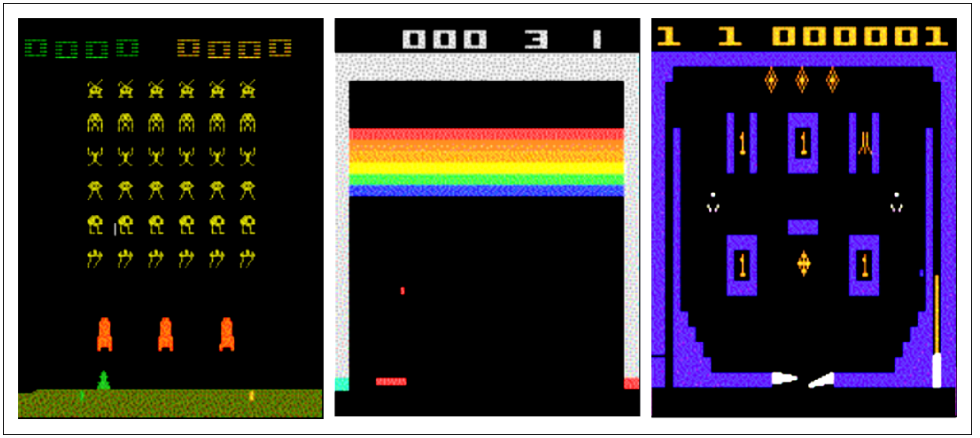


Abbildung 7.25 Die Spiele Space Invaders, Breakout und Video Pinball ...

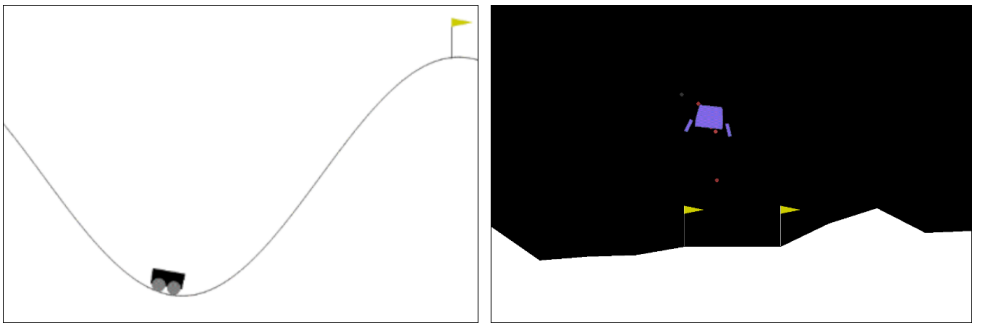


Abbildung 7.26 ... und die Physiksimulationen Mountain Car und Lunar Lander aus der OpenAi-Gym-Umgebung

© Rheinwerk Verlag, Bonn 2023

Die Schnittstellen der OpenAI-Gym-Umwelten sind vereinheitlicht. Daher können Sie einen Agenten, den Sie für eine bestimmte Umwelt entwickelt haben, im Prinzip auch auf andere Umwelten loslassen. OpenAI Gym ist bei Laien wie Profis weitverbreitet. Sie finden daher zahlreiche deutsch- und englischsprachige Tutorials und Beispielanwendungen im Netz.

7.12.2 Das Buch von Sutton und Barto

Wenn Sie es ganz genau wissen wollen, die englische Sprache gut beherrschen und sich auch von komplexeren mathematischen Ausdrücken nicht abschrecken lassen, können wir Ihnen das Buch »Reinforcement Learning: An Introduction« von Richard S. Sutton und Andrew G. Barto empfehlen. Das Buch bietet eine umfassende Darstellung und steht in der zweiten Auflage zum kostenlosen Download bereit. Alle Infos dazu finden Sie Anhang C, »Quellen und weiterführende Literatur«.

7.13 Zusammenfassung und Ausblick

Zum Abschluss dieses umfangreichen Kapitels wollen wir einen Vergleich mit dem menschlichen Erwerb von Fertigkeiten wagen und die Grenzen des hier vorgestellten Verfahrens aufzeigen.

7.13.1 Menschliches Lernen vs. Q-Learning

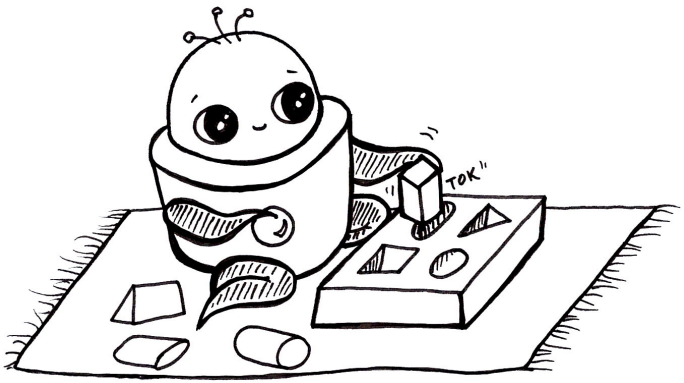
Bei den Recherchen und Codeexperimenten für dieses Kapitel sind uns einige Gemeinsamkeiten aufgefallen, die das Q-Learning mittels Q-Tabelle mit dem menschlichen Erwerb von Fähigkeiten aufweist. Der »Q-Lerner« hat zu Beginn des Lernprozesses keine Ahnung, wie er die ihm gestellte Aufgabe bewältigen soll. Daher bleibt ihm nichts anderes übrig, als herumzuprobieren. Ähnliches können wir bei einem Säugling beobachten, der versucht, einen Gegenstand zu greifen oder ein Steckspiel zu lösen. Erfolgserlebnisse bestärken den Säugling darin, die Lösung von gleichen oder ähnlichen Aufgaben zukünftig mit ähnlichen Bewegungsfolgen anzugehen wie jenen, die beim letzten Mal zum Erfolg geführt haben.

Je mehr ein Q-Lerner gelernt hat, umso weniger ist er auf zufällige Aktionen angewiesen – ebenso wie Menschen. Allerdings gilt für Q-Lerner und Mensch gleichermaßen: *Nur, weil eine Lösung funktioniert hat, muss sie nicht zwangsläufig die beste sein – schon gar nicht in einer Umwelt, die stetigem Wandel unterworfen ist.* »Das haben wir schon immer so gemacht« sollte als Begründung für ein bestimmtes Vorgehen allein nicht ausrei-

Persönliches Exemplar für Pit Noack

chen. Es kann in vielen Fällen nicht schaden, bewährte Strategien zu überdenken und gelegentlich etwas Neues zu probieren.

Im Bereich der Psychotherapie zeigt sich, dass in der Kindheit erlernte Problemlösungsstrategien zur Bewältigung des Lebens im Erwachsenenalter nicht immer geeignet sind. Und wir alle wissen, wie mühevoll es ist, einmal erlernte Gewohnheiten zu überwinden, auch wenn sich diese als unnütz oder gar schädlich erwiesen haben.



Einen deutlichen Unterschied zwischen einem lernenden Menschen und dem Q-Lerner besteht darin, dass Letzterer ohne ein Modell der Umwelt auskommt. Diese Modellfreiheit ist eine wesentliche Eigenschaft des Q-Learnings. Modellfreiheit bedeutet, dass ein Algorithmus keine Kenntnis über die Struktur der Umgebung benötigt, um Entscheidungen zu treffen. Er lernt direkt aus Erfahrung, ohne ein Modell der Umgebung zu bauen.

Wenn wir einen Stadtteil kennenlernen, merken wir uns die Anordnung der einzelnen Straßen und Gebäude zueinander. Wir erschaffen ein mentales Abbild des Stadtteils, eine Karte, wie unvollkommen diese auch immer sein mag. Der Q-Lerner erstellt keine Karte, sondern speichert lediglich die Beziehungen von Zuständen, Aktionen und Belohnungen.

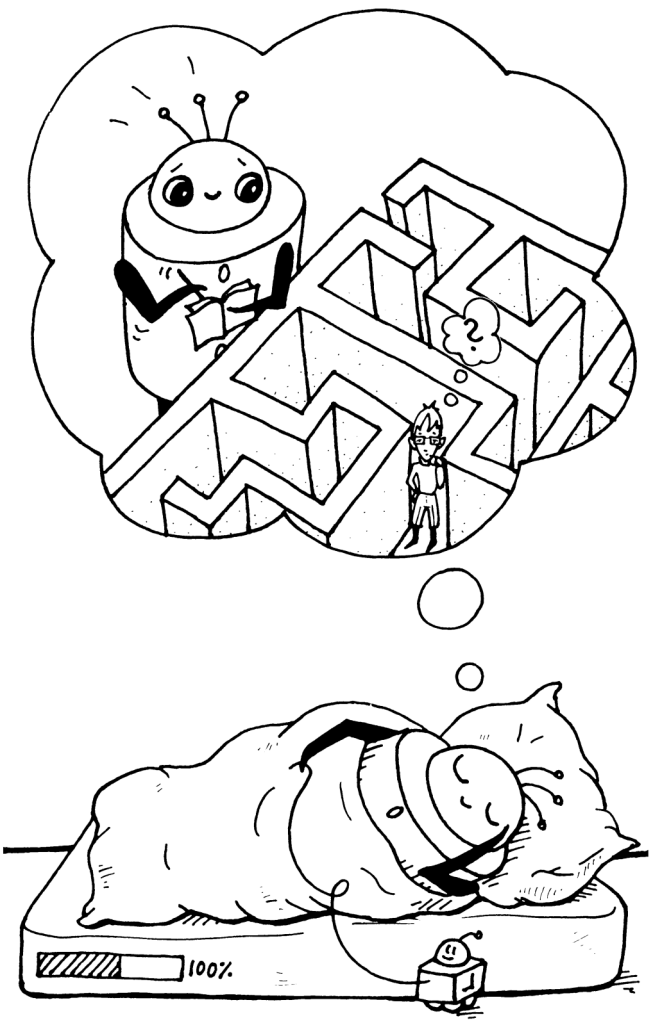
7.13.2 Die Grenzen des Verfahrens

Das hier vorgestellte Q-Learning mittels Q-Tabellen ist gut geeignet, um Lösungsstrategien für Aufgaben zu finden, die mit einer überschaubaren Anzahl von Zuständen und Aktionen auskommen. Wenn die Anzahl der möglichen Zustände und Aktionen zu groß ist, stößt das Verfahren an Grenzen. Stellen Sie sich vor, das Spielfeld ist 1.000 × 1.000 Zellen groß, es gibt vier Tageszeiten, und der Agent kann einen Spielstein aufheben oder

© Rheinwerk Verlag, Bonn 2023

ablegen. Schon haben Sie eine Q-Tabelle, die 8.000.000 Zustände und sechs Aktionen abbilden muss. Damit müssten 48.000.000 Q-Werte berechnet werden. Bei noch mehr Zuständen und Aktionen können Q-Tabellen den Arbeitsspeicher sprengen. Auch wenn das nicht passiert, kann es sehr lange dauern, eine optimale Strategie zu finden.

In solchen Fällen hat sich eine Erweiterung des Q-Learnings bewährt, das sogenannte *Deep Q-Learning*. Hier werden die Q-Werte nicht von Tabellen berechnet und gespeichert, sondern mittels neuronaler Netze. Über neuronale Netze erfahren Sie mehr in Kapitel 10, »Neuronale Netze I: Das Häschenproblem«. Abschnitt 13.5 geht etwas detaillierter auf das Thema Deep Q-Learning ein.



Persönliches Exemplar für Pit Noack



Hintergrund: Diskrete oder kontinuierliche Zustände und Zustandsräume

Daten lassen sich auf zwei Arten messen, speichern und darstellen: *diskret* und *kontinuierlich*. Diskrete Daten können nur eine abzählbare Anzahl von Werten annehmen, bei kontinuierlichen Daten gibt es diese Einschränkung nicht.

Nehmen wir als Beispiel die Größe von Schuhen und Füßen: So gibt es etwa Schuhe in den Größen 34 und 42. Im amerikanischen System sind auch Werte wie 10,5 oder 11,5 gebräuchlich. Nach Schuhen in Größe 34,76 oder 11,23 werden Sie allerdings vergeblich suchen. Ob Sie nun das europäische oder das amerikanische System verwenden: Immer haben Sie es mit einem diskreten System zu tun.

Messen Sie hingegen mit einem Maßband die Länge eines Fußes, kann das Messergebnis im Prinzip jeden beliebigen Wert annehmen, etwa 15,7 cm, 18,97 cm oder 21,633 cm. Die Zahl der Nachkommastellen hängt allein von der Genauigkeit Ihres Messverfahrens und der Entscheidung ab, wie viele Stellen Sie notieren wollen.

Das Q-Learning mittels Q-Tabelle lässt sich nur auf Umwelten anwenden, bei denen die Anzahl der Zustände begrenzt – also abzählbar, also diskret – ist. Wir sprechen in solchen Fällen auch von einem *diskreten Zustandsraum*. Falls wir aber mit dem Q-Lerner in Umwelten agieren wollen, die diese Bedingung nicht erfüllen, müssen wir die kontinuierlichen Zustände in diskrete Zustände übersetzen. Stellen wir uns ein Pendel vor, das zwischen A und B hin- und herschwingt. Eine Übersetzung dieser kontinuierlichen in diskrete Werte erreichen wir zum Beispiel, indem wir etwa die Entfernung von Punkt A auf zwei Nachkommastellen runden. Dieses Verfahren nennt sich *Quantisierung*. Mit ihr legen wir gleichsam ein diskretes Raster über den kontinuierlichen Wertebereich.

In vielen Fällen ist ein Raster mit gleichmäßigen Abständen allerdings nicht die beste Methode. Wenn die Aufgabe etwa darin besteht, eine Mondfähre sicher zu landen, sollten wir in der Nähe der Mondoberfläche ein feineres Raster auf die kontinuierlichen Werte für Höhe und Geschwindigkeit legen, als wenn wir noch einen Kilometer von der Oberfläche entfernt sind.

© Rheinwerk Verlag, Bonn 2023