mitp

V. Anton Spraul

THINK LIKE A PROGRAMMER

Typische Programmieraufgaben kreativ lösen am Beispiel von C++

Kapitel 1

Strategien zur Problemlösung

In diesem Buch geht es um das Lösen von Problemstellungen, aber was genau bedeutet das eigentlich? Wenn der Begriff in einer normalen Unterhaltung gebraucht wird, ist damit oft etwas ganz anderes gemeint als das, was wir hier meinen. Wenn Ihr 15 Jahre altes Auto blaue Rauchwölkchen aus dem Auspuff ausstößt, im Leerlauf ruckelt und sich der Benzinverbrauch drastisch erhöht hat, handelt es sich um eine Problemstellung, die sich mittels fahrzeugtechnischer Kenntnisse, einer Untersuchung des Automobils, Ersatzteilen und gebräuchlicher Werkzeuge lösen lässt. Wenn Sie nun Ihren Bekannten von diesem Problem berichten, könnte einer davon sagen: »Ach, du solltest das Auto gegen ein neues austauschen. Schon ist das Problem gelöst.« Allerdings wäre der Vorschlag Ihres Bekannten keine echte *Lösung* des Problems, sondern ein Weg, das Problem zu *umgehen*.

Problemstellungen beinhalten gewisse Randbedingungen, unantastbare Vorschriften hinsichtlich der Aufgabenstellung oder Bestimmungen, auf welche Art und Weise das Problem zu lösen ist. Im Fall des liegengebliebenen Fahrzeugs wäre eine dieser Bedingungen, dass Sie das alte Gefährt reparieren wollen und eben nicht ein neues Auto kaufen möchten. Weitere Randbedingungen wären beispielsweise die Gesamtkosten der Reparatur, deren Dauer oder die Auflage, dass nur für diese Instandsetzung kein neues Werkzeug beschafft werden soll.

Wenn eine Problemstellung mittels eines Programms gelöst wird, gibt es ebenfalls Randbedingungen. Dazu gehören üblicherweise die Programmiersprache, die Plattform (läuft das Programm auf einem PC, einem iPhone oder anderswo?), das Leistungsverhalten (manche Spiele müssen mindestens 30 mal pro Sekunde die Anzeige aktualisieren; einer geschäftlichen Anwendung steht möglicherweise nur eine begrenzte Zeitspanne zur Verfügung, um auf Benutzereingaben zu reagieren) oder der Speicherbedarf. Gelegentlich betreffen die Randbedingungen auch anderen Code, auf den zugegriffen werden soll. Vielleicht darf ein bestimmtes Programm keinen quelloffenen Code verwenden. Oder genau das Gegenteil: Möglicherweise darf ausschließlich quelloffener Code verwendet werden.

Für Programmierer können wir somit das *Lösen einer Problemstellung* wie folgt definieren: Schreiben eines eigenen Programms, das eine festgelegte Menge an Aufgaben erledigt und dabei allen genannten Randbedingungen genügt.

Programmieranfänger sind oft so darauf erpicht, den ersten Teil dieser Definition (ein Programm zu schreiben, um eine gewisse Aufgabe zu erledigen) in die Tat umzusetzen, dass sie dabei den zweiten Teil übersehen, nämlich den genannten

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278

Randbedingungen gerecht zu werden. Ich bezeichne ein solches Programm, das korrekte Ergebnisse zu liefern scheint, dabei aber eine oder mehrere der Randbedingungen verletzt, als *Kobayashi Maru*. Falls Ihnen dieser Name nichts sagt, sind Sie offenbar mit einem der Prüfsteine der Geek-Kultur, nämlich dem Film *Star Trek II: Der Zorn des Khan*, nicht hinreichend vertraut. In dem Film gibt es eine Rahmenhandlung, in der es um eine Prüfung angehender Offiziere der Sternenflottenakademie geht. Die Kadetten befinden sich dabei auf der Brücke eines simulierten Raumschiffs und leiten dort als Captain eine Mission, die zu einer unlösbaren Zwangslage führt. Die unschuldigen Menschen an Bord des schwer beschädigten Raumschiffs *Kobayashi Maru* werden sterben. Um diese Menschen zu retten, müsste eine Schlacht mit den Klingonen geschlagen werden, die zur sicheren Zerstörung des eigenen Schiffs des Captains führen würde. Der *Kobayashi Maru*-Test soll das Verhalten des Kadetten in einer ausweglosen Situation offenbaren. Es gibt keine Möglichkeit, den Test zu bestehen; jede Entscheidung führt zu einem verlustreichen Resultat. Gegen Ende des Films erfahren wir dann, dass Captain Kirk die Simulation manipuliert hat, um den Test tatsächlich bestehen zu können. Kirk war zwar schlau, aber er hat das Dilemma der *Kobayashi Maru* nicht gelöst, sondern nur umgangen.

Die Problemstellungen, denen Sie als Programmierer begegnen werden, sind glücklicherweise lösbar, aber viele Programmierer greifen dennoch auf Captain Kirks Ansatz zurück. In manchen Fällen geschieht dies versehentlich. (»Verflixt! Meine Lösung arbeitet nur mit weniger als 100 Datensätzen korrekt. Es muss aber mit beliebig vielen funktionieren. Das muss ich noch mal durchdenken.«) In anderen Fällen ist die Missachtung von Randbedingungen Absicht, etwa als List, um einen vom Vorgesetzten oder Ausbilder auferlegten Termin halten zu können. Und in wieder anderen Fällen weiß der Programmierer einfach nicht, wie er alle Bedingungen erfüllen kann. Das Schlimmste, was ich erlebt habe, war ein Student, der jemanden dafür bezahlte, ein Programm zu schreiben. Unabhängig von den Beweggründen ist ein Kobayashi Maru stets gewissenhaft zu vermeiden.

1.1 Klassische Rätsel

Im weiteren Verlauf des Buches wird Ihnen auffallen, dass sich die Einzelheiten des Quellcodes bei den verschiedenen Aufgabenstellungen zwar ändern, in den Lösungsansätzen jedoch gewisse Muster zum Vorschein kommen. Das ist deshalb von Bedeutung, weil es uns letztendlich erlaubt, beliebige Problemstellungen zuversichtlich anzugehen, und zwar unabhängig davon, ob wir mit dem Aufgabengebiet bereits umfangreiche Erfahrung haben oder nicht. Erfahrene Problemlöser sind in der Lage, schnell eine *Analogie* zu erkennen, eine Übereinstimmung zwischen einer bereits gelösten und einer ungelösten Aufgabe, die sich ausnutzen lässt. Wenn wir erkennen, dass eine Eigenschaft der Problemstellung A einer Eigenschaft der Problemstellung B entspricht und wir B bereits gelöst haben, haben wir wertvolle Erkenntnisse zur Lösung von A gewonnen.

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278

In diesem Abschnitt werden wir einige klassische Rätsel betrachten, die zwar nicht der Welt der Programmierung entstammen, uns aber Einsichten bescheren, die wir auf Problemstellungen der Programmierung anwenden können.

1.1.1 Fuchs, Gans und Getreidesack

Bei der ersten klassischen Aufgabenstellung handelt es sich um ein Rätsel, bei dem ein Bauer einen Fluss überqueren muss. Vermutlich ist es Ihnen bereits in der einen oder anderen Variante (z. B. Wolf, Ziege, Kohlkopf) begegnet.

Aufgabe: Wie kann der Fluss überquert werden?

Ein Bauer möchte zusammen mit einem Fuchs, einer Gans und einem Getreidesack einen Fluss überqueren. Der Bauer besitzt ein Ruderboot, das jedoch nur Platz für den Bauern und jeweils eines der drei Objekte bietet. Leider sind sowohl Fuchs als auch Gans sehr hungrig. Der Fuchs darf also nicht zusammen mit der Gans zurückgelassen werden, da er sie dann fressen würde. Auch die Gans darf nicht zusammen mit dem Getreidesack alleingelassen werden, weil sie in diesem Fall das Getreide frisst. Wie stellt der Bauer es an, alles zur anderen Uferseite zu transportieren?

In Abbildung 1.1 sind die Vorgaben des Rätsels dargestellt. Falls es Ihnen noch unbekannt ist, sollten Sie nun innehalten und ein paar Minuten lang probieren, es zu lösen. *Ist* es Ihnen jedoch bekannt, versuchen Sie, sich an die Lösung zu erinnern und ob Sie das Rätsel selbst lösen konnten.

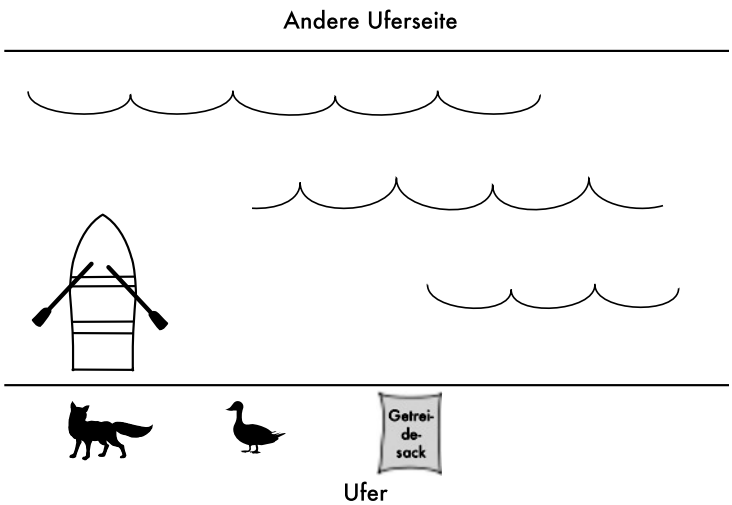

Abb. 1.1: Fuchs, Gans und Getreidesack. Das Boot kann jeweils nur eines der Objekte gleichzeitig transportieren. Weder Fuchs und Gans noch Gans und Getreidesack dürfen zusammen am selben Ufer zurückgelassen werden.

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278

Nur wenige Leute sind in der Lage, dieses Rätsel zu lösen, zumindest nicht ohne einen Tipp. Ich konnte es jedenfalls nicht. Normalerweise werden folgende Überlegungen angestellt: Da der Bauer jeweils nur eines der Objekte transportieren kann, muss er den Fluss mehrmals überqueren, um alles zur anderen Uferseite zu befördern. Falls der Bauer auf der ersten Fahrt den Fuchs mitnimmt, wäre die Gans mit dem Getreidesack allein und würde das Getreide auffressen. Nähme der Bauer zunächst den Getreidesack mit, blieben Fuchs und Gans zurück, und der Fuchs fräße die Gans. Folglich muss der Bauer als Erstes die Gans mitnehmen (siehe Abbildung 1.2).

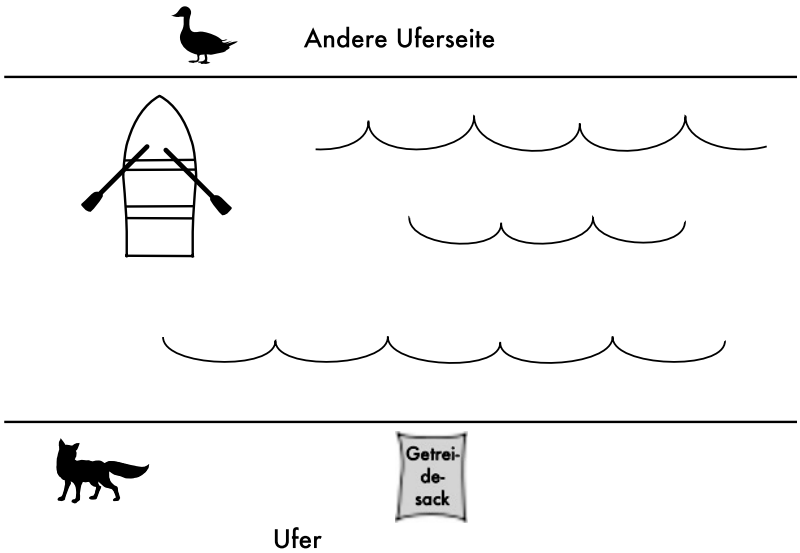

Abb. 1.2: Der erste Schritt zur Lösung des Fuchs-Gans-Getreidesack-Rätsels. Von hier aus scheinen jedoch alle weiteren Schritte zum Scheitern zu führen.

So weit, so gut. Aber bei der zweiten Fahrt muss der Bauer entweder den Fuchs oder den Getreidesack mitnehmen. Was er auch wählt, verbleibt bei der Gans auf der anderen Uferseite, wenn er zurückkehrt, um das letzte Objekt zu holen. Also bleiben entweder Fuchs und Gans oder aber Gans und Getreidesack zusammen zurück. Da beide Möglichkeiten nicht in Frage kommen, scheint die Aufgabe unlösbar.

Sollten Sie das Rätsel schon kennen, werden Sie sich wahrscheinlich an den entscheidenden Punkt der Lösung erinnern. Der Bauer nimmt bei der ersten Fahrt, wie oben erläutert, die Gans mit. Nehmen wir an, der Bauer transportiert bei der zweiten Fahrt den Fuchs. Statt nun aber den Fuchs bei der Gans zu belassen, *befördert er die Gans wieder zurück*. Der Bauer nimmt dann den Getreidesack mit, den er zusammen mit dem Fuchs auf der anderen Uferseite belässt, und kehrt dann

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278

zurück, um schließlich ein viertes Mal mit der Gans zum anderen Ufer zu rudern. Die vollständige Lösung ist in Abbildung 1.3 dargestellt.

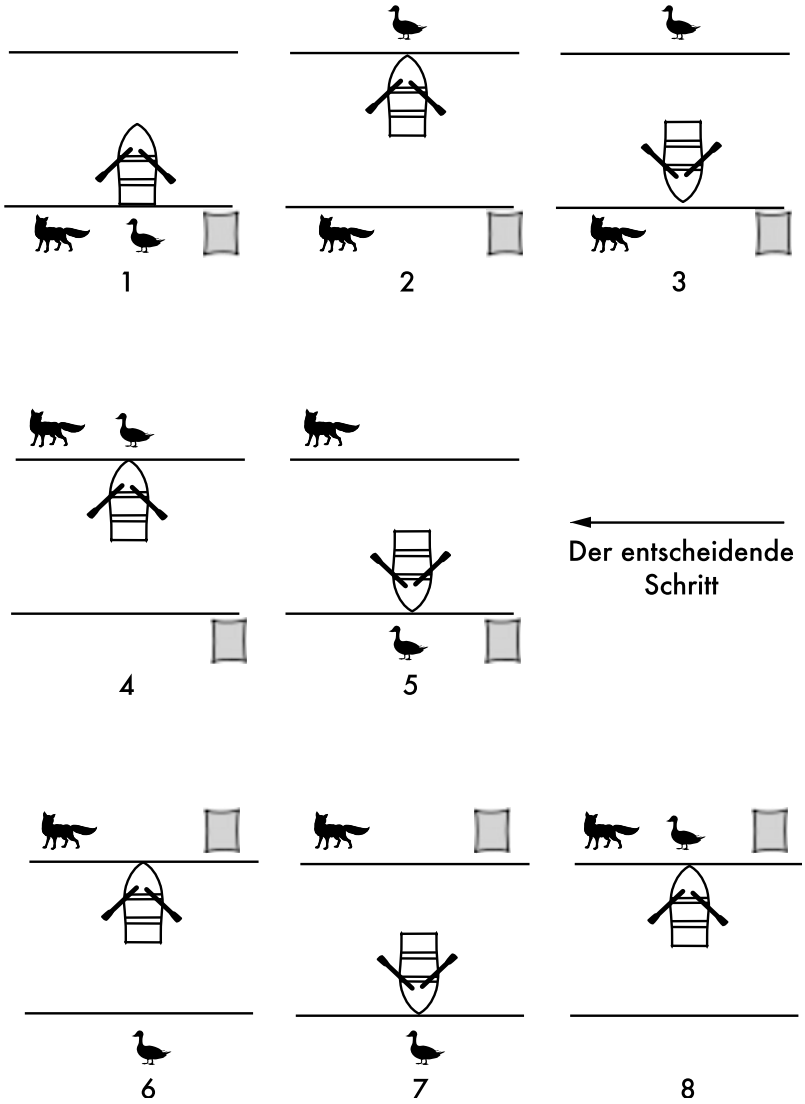

Abb. 1.3: Schrittweise Lösung des Fuchs-Gans-Getreidesack-Rätsels

Das Rätsel ist so schwierig, weil die meisten Leute gar nicht auf den Gedanken kommen, eines der Objekte wieder zurückzubefördern. Manche Leute sind sogar der Ansicht, dass das Rätsel unfair sei, und beschweren sich: »Du hast mit keinem Wort erwähnt, dass ich die Objekte zurücktransportieren darf!« Das stimmt zwar,

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278

aber es ist ebenso richtig, dass nichts in der Aufgabenstellung darauf hinweist, dass ein Rücktransport verboten ist.

Denken Sie doch einmal darüber nach, wie viel einfacher das Rätsel zu lösen wäre, wenn die Möglichkeit eines Rücktransports ausdrücklich erwähnt wäre: *Der Bauer besitzt ein Ruderboot,* **das zum Transport der Objekte in beide Richtungen verwendet werden kann***, jedoch nur Platz für den Bauern und jeweils eines der drei Objekte bietet*. Mit diesem offensichtlichen Hinweis würden viel mehr Leute die Lösung herausfinden. Hierdurch wird ein wichtiges Prinzip beim Problemlösen deutlich: Falls Sie sich nicht aller erlaubten Handlungsmöglichkeiten bewusst sind, können Sie das Problem möglicherweise nicht lösen. Wir bezeichnen diese Handlungsmöglichkeiten als *Operationen*. Durch die Aufzählung sämtlicher möglicher Operationen lassen sich viele Problemstellungen lösen, indem alle Kombinationen der Operationen ausprobiert werden, bis wir eine finden, die funktioniert. Oder allgemeiner ausgedrückt: Durch die Umformulierung einer Problemstellung in formaleren Begriffen lassen sich häufig Lösungen finden, die uns anderenfalls entgangen wären.

Lassen Sie uns für einen Augenblick vergessen, dass wir die Lösung bereits kennen, und versuchen wir, das vorliegende Rätsel formaler auszudrücken. Zunächst zählen wir die Randbedingungen auf. Die entscheidenden sind hier:

1. Der Bauer kann nur jeweils ein Objekt mit dem Boot transportieren.
2. Fuchs und Gans dürfen nicht zusammen am selben Ufer belassen werden.
3. Gans und Getreidesack dürfen nicht zusammen am selben Ufer belassen werden.

Diese Aufgabenstellung ist ein schönes Beispiel für die Bedeutung von Randbedingungen. Wenn irgendeine der Randbedingungen entfernt wird, ist das Rätsel simpel. Entfernen wir die erste Randbedingung, kann der Bauer einfach alle drei Objekte auf der ersten Fahrt mitnehmen. Selbst wenn er mit dem Boot nur zwei Objekte befördern dürfte, könnte er zunächst Fuchs und Getreidesack und nach der Rückkehr die Gans transportieren. Entfällt die zweite Randbedingung (während die anderen beiden weiterhin gelten), müssen wir nur ein wenig Vorsicht walten lassen. Es muss erst die Gans, dann der Fuchs und schließlich der Getreidesack ans andere Ufer verfrachtet werden. Wenn wir also eine der Randbedingungen übersehen oder ignorieren, gelangen wir letztendlich zu einem Kobayashi Maru.

Lassen Sie uns nun die Operationen aufzählen. Zur Formulierung der Operationen dieses Rätsels gibt es verschiedene Möglichkeiten. Wir könnten etwa eine ausführliche Liste der Handlungen erstellen, die wir glauben ausführen zu können:

1. Operation: Fuchs zur anderen Uferseite befördern.
2. Operation: Gans zur anderen Uferseite befördern.
3. Operation: Getreidesack zur anderen Uferseite befördern.

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278

Denken Sie jedoch daran, dass es das Ziel der formalen Umformulierung ist, Erkenntnisse über einen Lösungsweg zu gewinnen. Wenn wir das Rätsel nicht schon gelöst und die »verborgene« Operation »Rücktransport der Gans« entdeckt hätten, würden wir diese beim Erstellen der Liste der Handlungsmöglichkeiten nicht aufspüren. Stattdessen sollten wir versuchen, die Operationen in allgemeiner Form oder *parametrisiert* darzustellen:

1. Operation: Boot von einem Ufer zum anderen rudern.
2. Operation: Falls das Boot leer ist, mit einem Objekt vom Ufer beladen.
3. Operation: Falls das Boot nicht leer ist, Objekt entladen und am Ufer belassen.

Wenn man das Rätsel in den allgemeinsten Begriffen beschreibt, ermöglicht die zweite Liste von Operationen es uns, die Aufgabenstellung ohne ein Aha-Erlebnis hinsichtlich des Rücktransports der Gans zu lösen. Wenn wir eine Liste aller möglichen Abfolgen von Handlungen erstellen und diese jeweils beenden, wenn eine der Randbedingungen verletzt wird oder ein schon früher erreichter Zustand erreicht wird, werden wir irgendwann auf die Abfolge in Abbildung 1.3 stoßen und das Rätsel lösen. Die dem Rätsel innewohnende Schwierigkeit wird durch die formale Neuformulierung der Randbedingungen und Operationen umgangen.

Gewonnene Erkenntnisse

Welche Einsichten liefert uns das Fuchs-Gans-Getreidesack-Rätsel?

Die Neuformulierung der Problemstellung in einer formaleren Art und Weise ist eine ausgezeichnete Vorgehensweise, um Einblick in das Problem zu gewinnen. Viele Programmierer suchen den Kontakt zu anderen Programmierern, um eine Aufgabenstellung zu diskutieren. Nicht nur, weil die anderen Programmierer vielleicht die Lösung kennen, sondern auch, weil die ausdrückliche Artikulierung der Problemstellung oft neue und nützliche Einfälle auslöst. Die Neuformulierung der Aufgabenstellung ähnelt der Diskussion mit einem anderen Programmierer, nur dass Sie hier beide Rollen einnehmen.

Die weitergefasste Erkenntnis ist die, dass das Nachdenken über die Problemstellung genauso produktiv oder in einigen Fällen sogar produktiver als das Nachdenken über die Lösung ist. In vielen Fällen *ist* der richtige Lösungsansatz fast schon die Lösung.

1.1.2 Schiebepuzzles

Schiebepuzzles gibt es in verschiedenen Größen, für die es spezielle Lösungsverfahren gibt, wie Sie später noch sehen werden. Die folgende Aufgabenstellung bezieht sich auf ein Schiebepuzzle mit 3x3 Feldern.

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278

Aufgabe: Verschieben der 8

Ein Gitternetz mit 3x3 Feldern beherbergt 8 Kacheln, die von 1 bis 8 durchnummeriert sind, sowie ein leeres Feld. Eingangs befindet sich das Gitternetz in durcheinandergewürfeltem Zustand. Eine Kachel lässt sich in ein angrenzendes leeres Feld verschieben, wobei das ursprüngliche Feld der Kachel zum leeren Feld wird. Ziel ist es, die Kacheln so zu verschieben, dass sich das Gitternetz in sortiertem Zustand befindet, angefangen mit der Kachel 1 oben links.

Abbildung 1.4 zeigt das angestrebte Ziel dieser Aufgabenstellung. Falls Sie noch nie ein solches Schiebepuzzle ausprobiert haben, sollten Sie sich jetzt die Zeit dafür nehmen. Im Internet finden sich zwar eine Menge Schiebepuzzlesimulatoren, für unsere Zwecke ist es jedoch besser, Spielkarten oder Karteikarten zu verwenden, um damit auf einer Tischplatte zu spielen. In Abbildung 1.5 sehen Sie einen empfohlenen Ausgangszustand.

1	2	3
4	5	6
7	8	

Abb. 1.4: Der Zielzustand eines Schiebepuzzles mit 8 Kacheln. Das leere Quadrat repräsentiert das leere Feld, in das sich eine angrenzende Kachel verschieben lässt.

4	7	2
8	6	1
3	5	

Abb. 1.5: Ein spezieller Ausgangszustand des Schiebepuzzles

Dieses Puzzle unterscheidet sich deutlich vom Fuchs-Gans-Getreidesack-Rätsel. Dessen Schwierigkeit rührt daher, dass eine der möglichen Operationen übersehen wird. Das ist beim Schiebepuzzle nicht der Fall. Bei irgendeinem vorgegebenen Zustand gibt es bis zu vier Kacheln, die an das leere Feld angrenzen und in dieses verschoben werden können. Dadurch sind alle möglichen Operationen vollständig beschrieben.

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278

Die Schwierigkeit dieser Aufgabenstellung besteht in der langen Sequenz von Operationen, die zur Lösung erforderlich ist. Eine Abfolge von Verschiebeoperationen mag zwar einige Kacheln in die korrekte Zielposition bewegen, kann aber gleichzeitig andere Kacheln von ihrer Zielposition fortbefördern oder aber einige Kacheln näher an ihre Zielposition bringen, andere hingegen weiter von ihrer Zielposition entfernen. Aus diesem Grund ist es problematisch festzustellen, ob eine bestimmte Operation zu einem Fortschritt in Richtung Zielzustand führt. Wenn sich aber der Fortschritt nicht messen lässt, ist es schwierig, eine Strategie anzugeben. Viele Leute, die sich an einem Schiebepuzzle versuchen, bewegen die Kacheln einfach aufs Geratewohl herum, in der Hoffnung, auf einen Zustand zu treffen, von dem aus ein Weg zum Zielzustand erkennbar ist.

Gleichwohl gibt es Lösungsstrategien für Schiebepuzzles. Lassen Sie uns zur Veranschaulichung eines solchen Ansatzes ein Schiebepuzzle mit kleinerem Gitternetz, das zwar rechteckig, aber nicht quadratisch ist, betrachten.

Aufgabe: Verschieben der 5

Ein Gitternetz mit 2x3 Feldern beherbergt 5 Kacheln, die von 4 bis 8 durchnummeriert sind, sowie ein leeres Feld. Eingangs befindet sich das Gitternetz in durcheinandergewürfeltem Zustand. Eine Kachel lässt sich in ein angrenzendes leeres Feld verschieben, wobei das ursprüngliche Feld der Kachel zum leeren Feld wird. Ziel ist es, die Kacheln so zu verschieben, dass sich das Gitternetz in sortiertem Zustand befindet, angefangen mit der Kachel 4 oben links.

Sie werden bemerkt haben, dass die fünf Kacheln nun von 4 bis 8 statt von 1 bis 5 nummeriert sind. Der Grund dafür wird in Kürze klar werden.

Auch wenn es sich hier um dieselbe grundlegende Aufgabenstellung wie beim Verschieben der 8 handelt, gestaltet sich das Problem mit nur fünf Kacheln erheblich einfacher. Probieren Sie doch mal den in Abbildung 1.6 gezeigten Ausgangszustand aus.

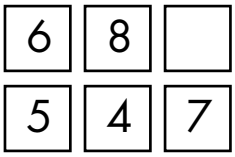

Abb. 1.6: Ausgangszustand für ein verkleinertes Schiebepuzzle mit 2x3 Feldern

Wenn Sie mit diesen Kacheln auch nur wenige Minuten herumspielen, werden Sie wahrscheinlich auf eine Lösung stoßen. Durch das Herumexperimentieren mit Schiebepuzzles, die nur wenige Kacheln besitzen, habe ich eine besondere Fähigkeit entwickelt. Es ist eine Fähigkeit in Verbindung mit einer Beobachtung, die gleich zur Sprache kommen wird, die ich zur Lösung aller Schiebepuzzles verwende.

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278

Ich bezeichne meine Vorgehensweise als »die Eisenbahn«. Sie beruht auf der Feststellung, dass ein geschlossener Ring von Kachelpositionen, der auch das leere Feld umfasst, eine Art »Eisenbahn« (mit den Kacheln als Waggons) bildet, die im Kreis herumfahren kann, ohne dass sich die relative Anordnung der Kacheln zueinander ändert. Abbildung 1.7 zeigt die kleinstmögliche Eisenbahn mit vier Kachelpositionen. Ausgehend von der ersten Anordnung kann Kachel 1 ins leere Feld verschoben werden, Kachel 2 nimmt die von Kachel 1 freigemachte Position ein, und schließlich wird Kachel 3 auf die freigewordene ehemalige Position von Kachel 2 verschoben. Kachel 1 grenzt nun wieder an das leere Feld an, sodass die Eisenbahn ihre Fahrt fortsetzen und somit beliebig auf der »Bahnstrecke« positioniert werden kann.

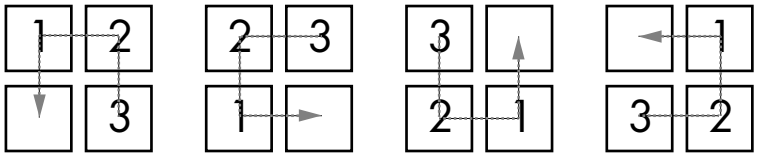

Abb. 1.7: Eine »Eisenbahn«, eine Folge von Kacheln, die am leeren Feld angrenzend beginnt und sich wie eine Reihe von Waggons durch das Schiebepuzzle schlängelt

Durch die Verwendung einer Eisenbahn können wir eine Sequenz von Kacheln verschieben und dabei deren relative Anordnung zueinander beibehalten. Wenden wir uns nun wieder dem ursprünglichen Gitternetz mit 2x3 Feldern (Abbildung 1.6) zu. Auch wenn noch keine der Kacheln die korrekte Zielposition einnimmt, besitzen doch einige der Kacheln schon den richtigen Nachbarn, neben dem sie sich im Zielzustand befinden sollen. Beispielsweise muss sich die 4 im Zielzustand über der 7 befinden, und derzeit sind die beiden schon Nachbarn. Wie Sie der Abbildung 1.8 unschwer entnehmen können, lässt sich eine Eisenbahn mit sechs Kachelpositionen dazu einsetzen, die Kacheln zur erwünschten Zielposition zu bewegen. Nachdem das erledigt ist, muss nur noch Kachel 8 nach links bewegt werden.

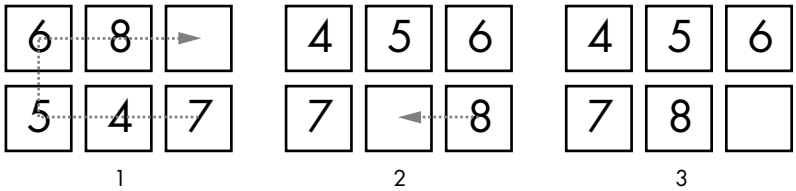

Abb. 1.8: Ausgehend von Zustand 1 sorgt die Fahrt der Eisenbahn zur zweiten Haltestelle dafür, dass wir zu Zustand 2 gelangen. Nun muss nur noch Kachel 8 nach links verschoben werden, um den Zielzustand 3 zu erreichen.

Und diese Vorgehensweise soll es uns ermöglichen, alle möglichen Schiebepuzzles zu lösen? Betrachten wir erneut das ursprüngliche Gitternetz mit 3x3 Feldern.

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278

Wir können eine Eisenbahn mit sechs Kachelpositionen verwenden, um die bereits benachbarten Kacheln 1 und 2 so zu positionieren, dass Kachel 2 sich neben Kachel 3 befindet (Abbildung 1.9).

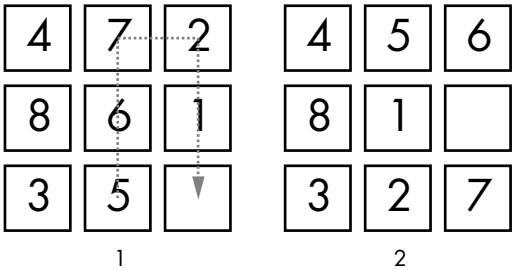

Abb. 1.9: Ausgehend von Zustand 1 fährt die Eisenbahn mehrere Haltestellen weiter, um Zustand 2 zu erreichen.

Die Kacheln 1, 2 und 3 befinden sich nun in aneinander angrenzenden Feldern. Mittels einer Eisenbahn mit acht Kachelpositionen können wir nun die Kacheln 1, 2 und 3 auf die korrekten Zielpositionen bewegen (Abbildung 1.10).

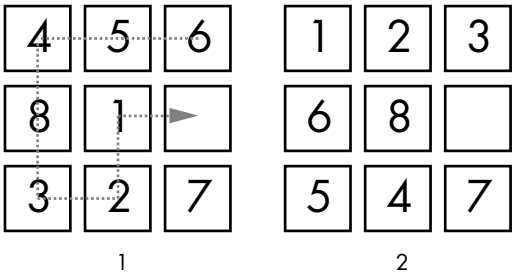

Abb. 1.10: Ausgehend von Zustand 1 fährt die Eisenbahn nach oben links, um die Kacheln 1, 2 und 3 auf den Zielpositionen zu platzieren.

Beachten Sie die Positionen der Kacheln 4 bis 8, die dem Zustand entsprechen, den ich in Abbildung 1.6 für ein 2x3-Gitternetz vorgeschlagen habe. Diese Beobachtung ist ganz entscheidend! Da sich die Kacheln 1, 2 und 3 nun an der richtigen Stelle befinden, können wir das verbleibende Gitternetz als ein separates, kleineres und einfacheres Schiebepuzzle betrachten und lösen. Beachten Sie hier, dass eine ganze Zeile oder Spalte die korrekten Werte enthalten muss, damit diese Methode funktioniert. Wenn die Kacheln 1 und 2 korrekt belegt sind, Kachel 3 jedoch einen falschen Wert enthält, ist es nicht möglich, diese Kachel durch eine andere zu ersetzen, ohne auch mindestens eine der beiden Kacheln 1 und 2 zu bewegen.

Dieses Verfahren lässt sich auch zum Lösen noch größerer Schiebepuzzles einsetzen. Üblicherweise besitzen Schiebepuzzles höchstens 15 Kacheln, verwenden

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278

also 4x4 Felder. Diese Puzzles lassen sich schrittweise lösen, indem zunächst die Kacheln 1 bis 4 auf die Zielposition bewegt werden, sodass ein 3x4 Felder großes Gitternetz verbleibt. Nun müssen die Kacheln der linken Spalte auf dieselbe Weise wie zuvor sortiert werden. Es handelt sich jetzt also um ein 3x3-Gitternetz. Dieses Problem haben wir mit der Aufgabe »Verschieben der 8« bereits gelöst.

Gewonnene Erkenntnisse

Welche Einsichten liefert uns das Schiebepuzzle?

Die Anzahl der möglichen Operationen ist so groß, dass es schwierig oder unmöglich ist, anhand des Ausgangszustands einen Lösungsweg anzugeben. Das hindert uns aber nicht daran, Strategien zu entwerfen oder Vorgehensweisen einzusetzen, um die Problemstellung systematisch zu lösen. Beim Lösen von Programmierungsaufgaben sehen wir uns manchmal Situationen gegenüber, in denen ein klarer Weg zur Programmierung der Lösung nicht absehbar ist, aber dieser Umstand darf niemals als Entschuldigung dafür herhalten, auf eine Planung und systematische Ansätze zu verzichten. Es ist allemal besser, eine Strategie zu entwickeln, als das Problem durch Versuch und Irrtum in Angriff zu nehmen.

Ich habe meine »Eisenbahn«-Technik beim Herumtüfteln mit kleinen Schiebepuzzles entwickelt. Beim Programmieren nutze ich oft eine ähnliche Vorgehensweise. Wenn ich mit einem hartnäckigen Problem konfrontiert werde, experimentiere ich mit einer vereinfachten Version davon herum. Diese Experimente führen oft zu wertvollen Erkenntnissen.

Darüber hinaus lässt sich hieraus lernen, dass sich eine Aufgabenstellung mitunter auf eine Art und Weise in Teilaufgaben zerlegen lässt, die nicht unmittelbar ersichtlich ist. Da das Verschieben einer Kachel nicht nur die Kachel selbst betrifft, sondern auch die danach möglichen Operationen beeinflusst, könnte man denken, dass ein Schiebepuzzle in einem einzigen Schritt gelöst werden muss, nicht etappenweise. Die Zeit für die Suche nach einer Möglichkeit, eine Problemstellung in Teilaufgaben zu zerlegen, ist für gewöhnlich gut investiert. Selbst wenn es Ihnen nicht gelingt, eine klare Zerlegung in Teilaufgaben zu finden, gelangen Sie möglicherweise zu anderen Einsichten in das Problem, die Ihnen dabei helfen, es zu lösen. Beim Problemlösen ein bestimmtes Ziel vor Augen zu haben, ist wahllosen Anstrengungen immer vorzuziehen, ob Sie dieses spezielle Ziel nun erreichen oder nicht.

1.1.3 Sudoku

Seit es in Tageszeitungen und Zeitschriften Sudokus gibt und diese auch auf Webseiten und Smartphones zu finden sind, erfreut sich dieses Spiel enormer Beliebtheit. Es gibt verschiedene Sudoku-Varianten, wir werden hier allerdings kurz die traditionelle Version betrachten.

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278

Aufgabe: Vervollständigen eines Sudoku-Quadrats

Einige der Felder eines Gitternetzes mit 9x9 Feldern enthalten einstellige Zahlen (von 1 bis 9). Der Spieler muss die leeren Felder unter Beachtung bestimmter Randbedingungen ausfüllen. In jeder Zeile und Spalte muss jede Ziffer genau einmal vorhanden sein; außerdem muss jedes der markierten 3x3 Felder großen Unterquadrate jede Ziffer von 1 bis 9 genau einmal enthalten.

Falls Sie schon mal ein Sudoku gelöst haben, verfügen Sie vermutlich bereits über verschiedene Strategien, um das Quadrat schnellstmöglich zu vervollständigen. Konzentrieren wir uns auf die entscheidende Strategie beim Start, indem wir uns das Beispiel-Sudoku in Abbildung 1.11 genauer ansehen.

	9	1		6		7		
				8	2		3	9
5		3				2		
			9	1	3		6	2
		2	4		6	8		
1	4		8	2	5			
		9				5		7
6	7		1	5				
		5		4		6	9	

Abb. 1.11: Ein einfaches Sudoku-Rätsel

Der Schwierigkeitsgrad von Sudoku-Rätseln variiert beträchtlich und wird durch die Anzahl der Felder bestimmt, die noch auszufüllen sind. Daran gemessen handelt es sich hier um ein sehr einfaches Rätsel. Da bereits 36 Felder eine Ziffer enthalten, sind lediglich 45 Felder auszufüllen, um das Sudoku zu vervollständigen. Nun stellt sich die Frage: Welches Feld sollen wir als erstes ausfüllen?

Rufen Sie sich die Randbedingungen ins Gedächtnis. Jede der neun Ziffern muss einmal in jeder Zeile, in jeder Spalte und in jedem der 3x3 Felder großen, durch einen fett gedruckten Rand markierten Unterquadrate vorkommen. Durch diese Spielregeln ist festgelegt, wo wir mit dem Ausfüllen anfangen sollten. Im 3x3 Felder großen Quadrat in der Mitte des Sudokus sind bereits acht der neun Felder ausgefüllt. Daher kann das leere Feld im Zentrum des Quadrats nur einen einzi-

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278

gen Wert besitzen, nämlich denjenigen, der in den anderen acht Feldern noch nicht vorhanden ist. Dabei handelt es sich um die 7, die wir nun dort eintragen.

Nachdem dieser Wert eingetragen ist, enthalten sieben der neun Felder der mittleren Spalte einen Wert, sodass nur noch zwei Werte verbleiben, die in dieser Spalte noch nicht vorkommen: 3 und 9. Die Randbedingung für die Spalte erlaubt es, eine der beiden Zahlen in jedes der noch leeren Felder einzutragen. Beachten Sie jedoch, dass die 3 bereits in der dritten Zeile vorkommt und die 9 schon in der siebten Zeile enthalten ist. Die Randbedingung für die Zeile legt daher fest, dass die 9 in die dritte Zeile und die 3 in die siebte Zeile der mittleren Spalte gehört. Diese Lösungsschritte sind in Abbildung 1.12 zusammengefasst.

	9	1		6		7		
				8	2		3	9
5		3				2		
			9	1	3		6	2
		2	4		6	8		
1	4		8	2	5			
		9				5		7
6	7		1	5				
		5		4		6	9	

1

	9	1		6		7		
				8	2		3	9
5		3				2		
			9	1	3		6	2
		2	4	7	6	8		
1	4		8	2	5			
		9				5		7
6	7		1	5				
		5		4		6	9	

2

	9	1		6		7		
				8	2		3	9
5		3		9		2		
			9	1	3		6	2
		2	4		6	8		
1	4		8	2	5			
		9		3		5		7
6	7		1	5				
		5		4		6	9	

3

Abb. 1.12: Die ersten Schritte beim Lösen des Sudoku-Rätsels

Wir wollen an dieser Stelle nicht das gesamte Sudoku lösen, aber diese ersten Schritte machen die wichtige Tatsache deutlich, dass wir nach Feldern suchen, die möglichst wenige Werte annehmen dürfen, idealerweise nur einen einzigen.

Gewonnene Erkenntnisse

Die wichtigste durch das Sudoku-Rätsel gewonnene Erkenntnis ist es, dass wir nach dem Teil der Problemstellung suchen müssen, der den meisten Randbedingungen unterliegt. Zwar sind es oft die Randbedingungen, die eine Aufgabenstellung überhaupt schwierig machen (denken Sie an das Fuchs-Gans-Getreidesack-Rätsel), sie können jedoch auch unsere Überlegungen zur Lösung vereinfachen, weil dadurch Auswahlmöglichkeiten eliminiert werden.

Obwohl wir uns in diesem Buch nicht weiter mit künstlicher Intelligenz beschäftigen werden, möchte ich Folgendes kurz erwähnen. Es gibt im Gebiet der künstlichen Intelligenz eine Regel zum Lösen von Problemstellungen bestimmten Typs, die auf der »Variablen mit den meisten Randbedingungen« beruht. Das bedeutet, dass Sie bei Aufgabenstellungen, in denen Sie versuchen, verschiedenen Variablen unterschiedliche Werte zuzuweisen, um die Randbedingungen zu erfüllen, mit derjenigen Variablen anfangen sollten, die den meisten Randbedingungen unter-

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278

liegt. Oder anders ausgedrückt: Beginnen Sie mit der Variablen, bei der die Anzahl möglicher Werte am niedrigsten ist.

Es folgt ein Beispiel für derartige Überlegungen. Stellen Sie sich eine Gruppe von Arbeitskollegen vor, die gemeinsam Mittagessen gehen möchten. Sie werden nun gebeten, ein Restaurant auszuwählen, das allen zusagt. Der Haken bei der Sache ist, dass jedes Gruppenmitglied der Gruppe als Ganzem eine Randbedingung auferlegt: Paula ist Vegetarierin, Thomas mag kein chinesisches Essen usw. Wenn Sie schnellstmöglich ein geeignetes Restaurant ausfindig machen möchten, sollten Sie zunächst denjenigen Kollegen befragen, der die schwerwiegendsten Einschränkungen mitbringt. Falls beispielsweise Robert gleich unter einer ganzen Reihe von Lebensmittelallergien leidet, wäre es sinnvoll, eine Liste der Restaurants zu erstellen, in denen er ohne Beschwerden speisen kann, anstatt sich um Thomas zu kümmern, dessen Abneigung gegen chinesisches Essen sich leicht lindern lässt.

Diese Technik lässt sich häufig auch auf Problemstellungen bei der Programmierung anwenden. Wenn ein Teil der Aufgabenstellung sehr vielen Randbedingungen genügen muss, sollten Sie Ihre Arbeit auf diesem Gebiet beginnen, denn Sie können hier Fortschritte erzielen, ohne sich Sorgen machen zu müssen, dass Sie Ihre Zeit mit Arbeiten verbringen, die später möglicherweise zunichtegemacht werden. Eine hiermit in Zusammenhang stehende Konsequenz ist es, mit dem Naheliegenden zu beginnen. Wenn Sie einen Teil der Problemstellung lösen können: Nur zu, erledigen Sie, was Sie können. Vielleicht entdecken Sie beim Betrachten Ihres eigenen Codes etwas, das Ihre Vorstellungskraft so anregt, dass Sie die verbleibende Aufgabe lösen können.

1.1.4 Das Quarrasi-Schloss

Die vorhergehenden Rätsel waren Ihnen möglicherweise bereits bekannt, doch das folgende werden Sie nicht kennen (sofern Sie dieses Buch nicht schon gelesen haben), denn ich habe es mir selbst ausgedacht. Lesen Sie es sorgfältig durch, denn der Wortlaut dieses Rätsels ist ein wenig kompliziert.

Aufgabe: Öffnen des außerirdischen Schlosses

Eine feindselige außerirdische Rasse, die Quarrasi, sind auf der Erde gelandet und haben Sie gefangen genommen. Es ist Ihnen gelungen, die Wächter zu überwältigen, obwohl diese von kolossaler Größe sind und Tentakel besitzen, allerdings müssen Sie zur Flucht aus dem (noch am Boden befindlichen) Raumschiff die äußerst solide Tür öffnen. Die Anleitung zum Öffnen der Tür ist merkwürdigerweise deutschsprachig, aber dennoch alles andere als ein Kinderspiel. Zum Öffnen der Tür müssen Sie drei stabförmige Kratzze entlang einer Führungsschiene vom rechten zum linken Rezeptor verschieben. Dieser befindet sich rund drei Meter entfernt am Ende der Tür.

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278

Nichts leichter als das! Allerdings müssen Sie es vermeiden, den Alarm auszulösen, der folgendermaßen funktioniert: Auf den Kratzzen befinden sich sternförmige Energiekristalle, sogenannte Quinicrys. Jeder Rezeptor besitzt vier Sensoren, die aufleuchten, wenn die Anzahl der Quinicrys in der jeweils darüber befindlichen Röhre eine gerade Zahl ist. Sobald die Anzahl der leuchtenden Sensoren exakt eins ist, wird der Alarm ausgelöst. Beachten Sie, dass die Alarmmechanismen der beiden Rezeptoren voneinander unabhängig sind. Es ist daher nicht möglich, dass bei beiden Rezeptoren gleichzeitig genau ein Sensor aufleuchtet. Die gute Nachricht lautet, dass sich beide Alarmmechanismen stummschalten lassen, wenn ein entsprechender Knopf gedrückt wird. Wenn Sie beide Stummschalter gleichzeitig betätigen könnten, wäre die Aufgabe leicht zu lösen; das können Sie aber nicht, denn Sie verfügen nur über kurze Menschenarme und nicht über die langen Tentakel der Quarrasi.

Nachdem Sie nun all dies wissen: Wie verschieben Sie die Kratzze zum Öffnen der Tür, ohne einen der beiden Alarme auszulösen?

Abbildung 1.13 zeigt den Ausgangszustand, in dem sich alle drei Kratzze im rechten Rezeptor befinden. Zur Verdeutlichung zeigt Abbildung 1.14, wie man es nicht machen sollte: Das Verschieben des obersten Kratzzes in den linken Rezeptor löst im rechten Rezeptor den Alarm aus. Nun denken Sie vielleicht, dass man den Alarm doch stummschalten könnte, aber wenn Sie soeben den oberen Kratzz in den linken Rezeptor verschoben haben, sind Sie drei Meter vom Stummschalter des rechten Rezeptors entfernt.

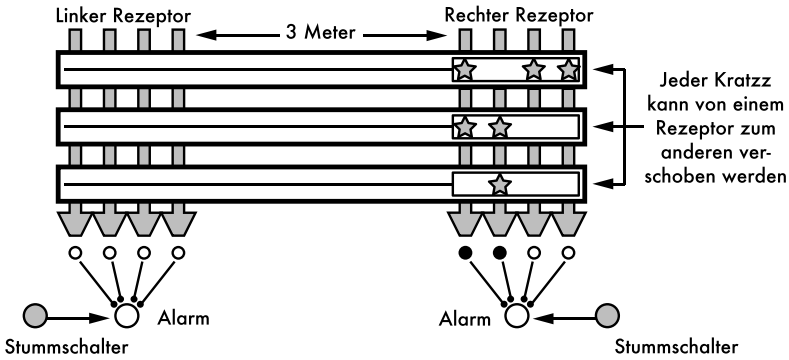

Abb. 1.13: Der Ausgangszustand des Quarrasi-Schlosses. Sie müssen die drei Kratzz-Stäbe, die sich momentan im rechten Rezeptor befinden, in den linken Rezeptor verschieben, ohne dabei einen der beiden Alarmmechanismen auszulösen. Ein Sensor leuchtet auf, wenn sich in der darüber befindlichen Röhre eine gerade Anzahl sternförmiger Quinicrys befindet, und der Alarm wird ausgelöst, wenn genau einer der Sensoren leuchtet. Durch den Stummschalter lässt sich das Ertönen des Alarms unterdrücken, jedoch nur bei dem Rezeptor, vor dem Sie sich gerade befinden.

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278

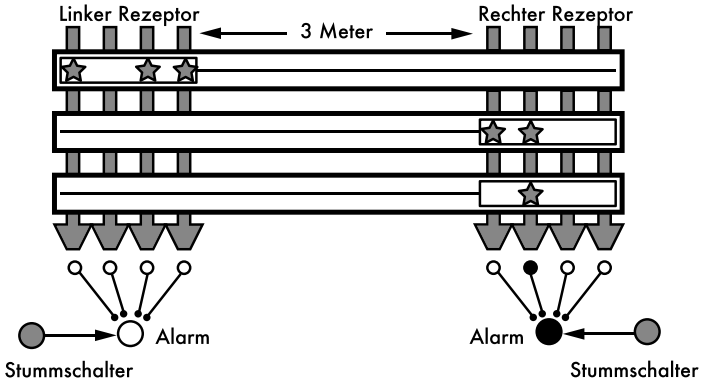

Abb. 1.14: Das Quarrasi-Schloss im Alarmzustand. Nachdem Sie den oberen Kratzz in den linken Rezeptor verschoben haben, ist der Stummschalter des rechten Rezeptors außer Reichweite. Der zweite Sensor des rechten Rezeptors leuchtet, da sich in der Röhre über ihm eine gerade Anzahl Quinicrys befindet, und der Alarm wird ausgelöst, wenn genau ein Sensor leuchtet.

Nehmen Sie sich einen Moment Zeit, um diese Problemstellung zu untersuchen, und versuchen Sie, eine Lösung zu finden, bevor Sie fortfahren. Je nach Standpunkt ist diese Aufgabenstellung gar nicht so schwer, wie es aussieht. Ernsthaft, denken Sie darüber nach, bevor Sie weiterlesen!

Haben Sie Überlegungen angestellt? Konnten Sie eine Lösung finden? Es gibt hier zwei mögliche Lösungswege. Einer davon erfolgt durch Versuch und Irrtum: systematisches Ausprobieren der verschiedenen Kratzzverschiebungen und jeweils einen Schritt zurückgehen, wenn ein Alarm ausgelöst wird, bis Sie eine Folge von Verschiebungen finden, die erfolgreich ist.

Der zweite Weg ist die Erkenntnis, dass es sich bei diesem Rätsel um einen Trick handelt. Falls Sie den Trick noch immer nicht entdecken können: Es handelt sich um die Problemstellung des Fuchs-Gans-Getreidesack-Rätsels in einer aufwendigen Verkleidung. Auch wenn die Regeln für das Auslösen des Alarms sehr allgemein formuliert sind, gibt es doch nur einige wenige mögliche Kombinationen für dieses spezielle Schloss. Bei nur drei Kratzzen müssen wir lediglich wissen, welche der Kratzzkombinationen in einem Rezeptor akzeptabel sind. Wenn wir die drei Kratzze als *Oben*, *Mitte* und *Unten* bezeichnen, lösen die Kombinationen Oben/Mitte und Mitte/Unten den Alarm aus. Wenn wir nun *Oben* in *Fuchs*, *Mitte* in *Gans* und *Unten* in *Getreidesack* umbenennen, sind die problematischen Kombinationen mit denen des anderen Rätsels identisch, nämlich Fuchs/Gans und Gans/Getreidesack.

Das Rätsel lässt sich daher auf dieselbe Weise wie das Fuchs-Gans-Getreidesack-Rätsel lösen. Wir verschieben zunächst den mittleren Kratzz (Gans) in den linken Rezeptor. Danach verschieben wir den oberen Kratzz (Fuchs) in den linken Rezep-

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278

tor und betätigen den linken Stummschalter, während der obere Kratzz (Fuchs) seinen Platz einnimmt. Nun schieben wir den mittleren Kratzz (Gans) zurück in den rechten Rezeptor, dann den unteren Kratzz (Getreidesack) nach links und schließlich den mittleren (Gans) erneut nach links, womit das Schloss geöffnet ist.

Gewonnene Erkenntnisse

Die wichtigste Erkenntnis ist, von welch großer Bedeutung das Erkennen von Analogien ist. Wir sehen hier, dass das Quarrasi-Schloss mit dem Fuchs-Gans-Getreidesack-Rätsel korrespondiert. Wenn wir diese Analogie rechtzeitig erkennen, können wir uns einen Großteil der Arbeit zur Lösung des Problems ersparen, indem wir die Lösung des bereits gelösten Problems übersetzen, anstatt eine neue Lösung zu suchen. Die meisten Analogien beim Problemlösen sind nicht so unmittelbar, sie werden aber immer häufiger auftauchen.

Wenn es Ihnen Schwierigkeiten bereitet hat, die Analogie zwischen der vorliegenden Problemstellung und dem Fuchs-Gans-Getreidesack-Rätsel wahrzunehmen, so liegt dies daran, dass ich absichtlich möglichst viele unerhebliche Details hinzugefügt habe. Die Rahmenhandlung des Quarrasi-Schloss-Rätsels ist völlig irrelevant, ebenso, wie es die Bezeichnungen für die außerirdischen Technologien sind, die nur dazu dienen, das Gefühl der Fremdartigkeit zu steigern. Und auch das Verfahren, eine gerade und ungerade Anzahl beim Auslösen des Alarms einzusetzen, macht die Sache komplizierter, als sie eigentlich ist. Wenn Sie sich die Positionen der Quinicrys genauer ansehen, werden Sie feststellen, dass oberer und unterer Kratzz Gegenstücke voneinander sind und sich daher beim Auslösen des Alarms überhaupt nicht beeinflussen. Der mittlere Kratzz allerdings wechselwirkt mit den anderen beiden.

Machen Sie sich also keine Sorgen, wenn Sie die Analogie nicht erkannt haben. Wenn Sie sich beim Aufspüren von Analogien in Alarmbereitschaft versetzen, werden Sie diese auch eher erkennen.

1.2 Gängige Verfahren zur Problemlösung

Die erläuterten Beispiele demonstrieren viele der entscheidenden Techniken, die beim Problemlösen eingesetzt werden. Im verbleibenden Teil des Buches werden wir spezielle Problemstellungen bei der Programmierung betrachten und herausfinden, wie sich diese lösen lassen, zunächst jedoch benötigen wir einige allgemeine Vorgehensweisen und Prinzipien. Wie Sie sehen werden, gibt es in manchen Aufgabengebieten spezifische Techniken, aber die nachstehenden Regeln treffen auf fast alle Situationen zu. Wenn Sie diese Regeln zu einem üblichen Bestandteil Ihrer Problemlösungsversuche machen, verfügen Sie immer über eine Methode, um ein Problem anzugehen.

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278

1.2.1 Niemals ohne Plan

Hierbei handelt es sich vermutlich um die wichtigste Regel überhaupt. Sie sollten stets einen Plan verfolgen, statt in blinden Aktionismus zu verfallen.

Mittlerweile dürfte Ihnen klar geworden sein, dass es immer möglich ist, einen Plan zu verfolgen. Wenn Sie eine Problemstellung nicht schon im Kopf gelöst haben, können Sie natürlich auch noch keinen Plan aufstellen, wie eine Lösung als Code umzusetzen ist – das kommt später. Sie sollten jedoch schon von Anfang an einen Plan besitzen, wie Sie zu einer Lösung kommen wollen.

Fairerweise muss gesagt werden, dass es erforderlich sein kann, den Plan während dessen Ausführung ändern zu müssen oder ihn zu verwerfen und einen neuen auszuhecken. Warum ist diese Regel dann überhaupt so wichtig? General Dwight D. Eisenhower ist für den Ausspruch »Ich habe stets festgestellt, dass Pläne nutzlos sind, aber eine Planung ist unverzichtbar« bekannt. Er wollte damit sagen, dass Schlachten so chaotisch sind, dass es unmöglich ist, alles vorherzusehen, was geschehen kann, und für alles eine passende Lösung bereitzuhalten. In diesem Sinne also sind Pläne auf dem Schlachtfeld wertlos (ein anderer Feldherr, der Preuße Helmuth von Moltke, sagte bekanntlich: »Kein Plan übersteht den ersten Kontakt mit dem Feind.«). Dennoch kommt keine Armee ohne Planung und Organisation aus. Durch Planung lernt ein General die Fähigkeiten seiner Armee kennen, erfährt, wie die einzelnen Abteilungen der Armee miteinander zusammenarbeiten, usw.

In gleicher Weise benötigen Sie einen Plan zum Lösen eines Problems. Er mag den ersten Kontakt mit dem Feind vielleicht nicht überstehen (möglicherweise muss er verworfen werden, sobald Sie damit anfangen, Code in Ihren Texteditor einzutippen), aber Sie müssen über einen Plan verfügen.

Ohne einen Plan hoffen Sie bestenfalls auf einen Glücksfall, dem Pendant zu den wahllos tippenden Affen, die eines der Werke Shakespeares erzeugen. Glücksfälle sind selten, und selbst in den wenigen Glücksfällen, die es tatsächlich gibt, benötigen Sie einen Plan. Den meisten Menschen ist die Geschichte von der Entdeckung des Penicillins bekannt: Der Forscher Alexander Fleming übersah es eines Abends, eine Petrischale zu verschließen, und entdeckte am nächsten Morgen, dass ein Schimmelpilz die Vermehrung der Bakterien in der Schale gehemmt hatte. Aber Fleming saß nicht herum und wartete auf einen Glücksfall; er hat sorgfältig und planvoll experimentiert und so die Bedeutung seiner Beobachtung in der Petrischale erkannt. (Wenn ich eines Morgens Schimmel auf irgendetwas entdecke, was ich am vorhergehenden Abend vergessen habe, wird das wohl kaum einen bedeutsamen Beitrag zur Wissenschaft leisten.)

Eine Planung erlaubt es Ihnen außerdem, Zwischenziele festzulegen und zu erreichen. Ohne Plan haben Sie nur ein einziges Ziel: das gesamte Problem zu lösen. Solange das Problem noch nicht gelöst ist, werden Sie nicht das Gefühl haben,

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278

irgendetwas erreicht zu haben. Wie Sie vermutlich schon selbst erfahren haben, sind viele Programme nicht besonders nützlich, wenn sie nicht schon kurz vor der Vollendung stehen. Nur auf das endgültige Ziel hinzuarbeiten, führt daher unausweichlich zu Frustration, und es gibt bis zum Schluss keine positive Verstärkung Ihrer Anstrengungen. Wenn Sie jedoch einen Plan mit einer Reihe kleinerer Zwischenziele (selbst wenn diese das Hauptproblem vielleicht kaum berühren) aufstellen, werden Sie merkliche Fortschritte auf dem Weg zur endgültigen Lösung machen und das Gefühl haben, dass Sie Ihre Zeit sinnvoll verbracht haben. Am Ende eines jeden Arbeitsabschnitts können Sie einen der Punkte auf Ihrem Plan abhaken und die Zuversicht gewinnen, dass Sie der Lösung näherkommen, statt zunehmend frustrierter zu werden.

1.2.2 Umformulieren der Aufgabenstellung

Wie insbesondere das Fuchs-Gans-Getreidesack-Rätsel zeigt, kann die Umformulierung der Problemstellung wertvolle Einsichten bringen. Eine Problemstellung, die äußerst schwierig erscheint, sieht in manchen Fällen ganz einfach aus, wenn sie anders formuliert oder mittels anderer Begrifflichkeiten beschrieben wird. Die Umformulierung einer Aufgabenstellung gleicht dem Umkreisen eines Hügels, der erstiegen werden soll. Warum nicht nach der einfachsten Aufstiegsmöglichkeit suchen, bevor Sie mit dem Klettern anfangen? Manchmal zeigt uns eine Umformulierung auch, dass das eigentliche Ziel gar nicht dasjenige ist, das wir ursprünglich im Auge hatten. Ich habe von einer Großmutter gelesen, die auf ihre kleine Enkeltochter aufpasste und dabei strickte. Um ihre Strickarbeiten in Ruhe erledigen zu können, setzte sie das Kind neben sich in einen Laufstall. Doch dem Kind gefiel es im Laufstall gar nicht und es weinte unaufhörlich. Die Großmutter versuchte, dem Kind den Laufstall mit allerlei Spielzeug zu versüßen, bis ihr der Gedanke kam, dass der Laufstall an sich das Mittel zum Zweck ist. Das Ziel der Großmutter war es, ungestört stricken zu können. Die Lösung: Das Kind spielt fröhlich auf dem Teppich, während die Großmutter im Laufstall sitzt und strickt. Die Umformulierung kann ein mächtiges Werkzeug sein, doch viele Programmierer nutzen es nicht, weil es nicht unmittelbar mit dem Schreiben von Code oder auch nur dem Entwerfen einer Lösung zu tun hat. Hierbei handelt es sich um einen weiteren Grund dafür, dass ein Plan unabdingbar ist. Ohne Plan besteht das einzige Ziel darin, zu funktionierendem Code zu gelangen, und das Umformulieren nimmt Zeit in Anspruch, die zum Schreiben von Code genutzt werden könnte. Gibt es einen Plan, können Sie »Formales Umformulieren der Problemstellung« als ersten Schritt eintragen – und schon zählt die Umformulierung offiziell als Fortschritt.

Selbst wenn die Umformulierung nicht unmittelbar zu neuen Einsichten führt, kann sie doch auf andere Weise hilfreich sein. Wenn Ihnen beispielsweise (von einem Vorgesetzten oder einem Ausbilder) eine Problemstellung zugewiesen wird, können Sie der betreffenden Person Ihre Umformulierung vorlegen und

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278

sich dadurch bestätigen lassen, dass Sie die Aufgabenstellung richtig verstanden haben. Die Umformulierung einer Problemstellung kann außerdem auch ein notwendiger Vorbereitungsschritt zur Verwendung anderer gängiger Techniken sein, wie etwa Reduktion oder Aufgliederung eines Problems.

Allgemeiner betrachtet lassen sich durch eine Umformulierung ganze Aufgabengebiete umgestalten. Bei der Vorgehensweise, die ich für rekursive Lösungen einsetze (dazu später mehr), handelt es sich um eine Methode zum Umformulieren rekursiver Problemstellungen, sodass ich diese genauso wie iterative behandeln kann.

1.2.3 Zerlegung in Teilaufgaben

Findet man einen Weg, um die Problemstellung in mehrere Schritte oder Phasen aufzuteilen, kann dies das Problem erheblich vereinfachen. Lässt sich eine Aufgabenstellung in zwei Teile zerlegen, könnte man denken, dass jeder Teil halb so schwer zu lösen wäre wie die gesamte Aufgabe, aber für gewöhnlich ist es sogar noch einfacher.

Dazu eine Analogie, die Ihnen bekannt sein dürfte, wenn Sie sich schon mit gebräuchlichen Sortieralgorithmen beschäftigt haben. Nehmen wir an, Sie möchten 100 Karteikarten in alphabetischer Reihenfolge in einen Karteikasten einsortieren. Ihre grundlegende Sortiermethode dabei ist das, was als *Insertionsort* (englisch *insertion*, Einfügen, und *to sort*, sortieren) bezeichnet wird: Sie nehmen eine beliebige Karteikarte und legen diese im Karteikasten ab. Die nächste Karteikarte legen Sie dann an der korrekten Position relativ zu der ersten Karteikarte ab und fahren damit fort, die jeweils neue Karteikarte an der richtigen Stelle relativ zu den anderen Karteikarten einzufügen, sodass die Karten im Karteikasten zu jedem Zeitpunkt alphabetisch sortiert sind. Nun stellen Sie sich vor, dass zuvor jemand die Karteikarten in vier ungefähr gleich große Gruppen, A–F, G–M, N–S und T–Z, aufgeteilt hat und Sie auffordert, die vier Gruppen einzeln zu sortieren und dann nacheinander im Karteikasten abzulegen.

Wenn jede Gruppe aus etwa 25 Karteikarten besteht, könnte man meinen, dass das Sortieren von 4 mal 25 Karteikarten in etwa genauso viel Arbeit macht wie das Sortieren einer einzigen Gruppe mit 100 Karteikarten. Tatsächlich ist es aber bedeutend *weniger* Arbeit, denn die erforderliche Zeit zum Einsortieren einer Karteikarte nimmt mit der Anzahl der bereits vorhandenen Karteikarten zu, schließlich müssen Sie immer im ganzen Karteikasten nachsehen, wo eine Karte hingehört. (Wenn Sie daran zweifeln, denken Sie an eine Extremversion; vergleichen Sie die Vorstellung, 50 Gruppen von 2 Karteikarten zu sortieren, was Ihnen vermutlich in weniger als einer Minute gelingt, mit dem Sortieren einer einzigen Gruppe von 100 Karteikarten.)

In gleicher Weise kann die Aufgliederung einer Problemstellung den Schwierigkeitsgrad oft um eine ganze Größenordnung verringern. Verschiedene Program-

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278

miertechniken zu kombinieren, ist allerdings sehr viel kniffliger als die alleinige Verwendung einer Programmtechnik. Ein Codeabschnitt beispielsweise, in dem sich eine Reihe von `if`-Anweisungen innerhalb einer `while`-Schleife befinden, die ihrerseits in eine `for`-Schleife eingebettet ist, wird schwieriger zu programmieren – und zu verstehen – sein, als ein Codeabschnitt, in dem dieselben Kontrollstrukturen hintereinander stehen.

Wir werden im nächsten Kapitel spezielle Verfahren zur Aufgliederung von Problemstellungen betrachten, aber Sie sollten sich dieser Möglichkeit stets bewusst sein. Denken Sie daran, dass bei manchen Aufgabenstellungen, wie etwa im Fall des Schiebepuzzles, die mögliche Unterteilung der Problemstellung oft verborgen ist. Wie Sie bald sehen werden, liefert gelegentlich auch die Reduktion einer Problemstellung, die Zurückführung auf ein anderes Problem, eine mögliche Unterteilung.

1.2.4 Mit Bekanntem anfangen

Erstmaligen Romanautoren wird oft »Schreibe über das, was du kennst« geraten. Das bedeutet nicht, dass Schriftsteller nur Werke über Ereignisse und Personen schreiben sollen, die sie in ihrem eigenem Leben unmittelbar selbst beobachtet haben. Wäre das der Fall, gäbe es weder Fantasy- noch historische Romane und auch so manch anderes weit verbreitete Genre nicht. Gemeint ist vielmehr Folgendes: Je weiter sich ein Autor von den eigenen Erfahrungen entfernt, umso schwieriger kann es werden, darüber zu schreiben.

In ähnlicher Weise sollten Sie beim Programmieren mit Bekanntem anfangen und sich von dort aus weiter vorarbeiten. Sobald Sie beispielsweise eine Aufgabe in Teile zerlegt haben, können Sie weitermachen und alle Programmteile vollenden, von denen Sie bereits wissen, wie sie zu programmieren sind. Eine funktionierende Teillösung kann dabei Anregungen liefern, die zur vollständigen Lösung beitragen. Ständiges Thema beim Lösen von Problemen ist es, wie Sie vielleicht bemerkt haben, brauchbare Fortschritte zu erzielen, um die Zuversicht aufzubauen, dass Sie letzten Endes die Aufgabenstellung vollständig erledigen werden. Wenn Sie mit Bekanntem anfangen, erlangen Sie Zutrauen und den erforderlichen Schwung, der zum Ziel führt.

Der Leitsatz »Mit Bekanntem anfangen« findet auch in solchen Fällen Anwendung, in denen Sie die Problemstellung nicht aufgegliedert haben. Stellen Sie sich vor, jemand hätte eine Liste mit sämtlichen Fähigkeiten beim Programmieren erstellt: eine C++-Klasse schreiben, eine Liste von Zahlen sortieren, den größten Wert in einer verketteten Liste finden usw. Zu jedem Zeitpunkt Ihrer Fortentwicklung als Programmierer wird es in dieser Liste Fähigkeiten geben, die Sie gut beherrschen, andere Fertigkeiten, zu denen Sie nur mit Anstrengung imstande sind, und weitere, die Ihnen noch gänzlich unbekannt sind. Eine bestimmte Problemstellung ist möglicherweise mit den Ihnen eigenen Fähigkeiten vollständig lösbar – oder auch nicht. Sie sollten aber dem vorliegenden Problem unter Ver-

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278

wendung der Ihnen bereits zur Verfügung stehenden Fähigkeiten vollständig auf den Grund gehen, bevor Sie sich nach anderen Lösungswegen umsehen. Wenn wir uns die Fähigkeiten beim Programmieren als Werkzeuge und eine Problemstellung bei der Programmierung als Heimwerkeraufgabe vorstellen, dann sollten Sie versuchen, die Reparatur mit den in Ihrer Garage bereits vorhandenen Werkzeugen durchzuführen, bevor Sie den Baumarkt aufsuchen.

Diese Vorgehensweise entspricht den bereits erläuterten Prinzipien. Sie folgt einem Plan und bringt Ordnung in unsere Bemühungen. Wenn wir die Untersuchung einer Problemstellung mit der Anwendung der uns bereits zur Verfügung stehenden Fähigkeiten beginnen, werden wir möglicherweise mehr über die Aufgabe und deren endgültige Lösung erfahren.

1.2.5 Reduktion der Problemstellung

Wenn Sie einer Problemstellung gegenüberstehen, die Sie nicht lösen können, ermöglicht es dieses Vorgehen, den Umfang des Problems durch Hinzufügen oder Entfernen von Randbedingungen zu reduzieren, sodass Sie zu einer Aufgabenstellung gelangen, für die Sie eine Lösung kennen. Später in diesem Kapitel werden Sie dieses Verfahren in Aktion sehen, doch zunächst ein einfaches Beispiel. Angenommen, Sie erhalten eine Reihe von Koordinaten im dreidimensionalen Raum. Sie sollen die Koordinaten finden, die einander am nächsten liegen. Falls Ihnen nicht sofort klar ist, wie das zu lösen ist: Es gibt mehrere Möglichkeiten, das Problem auf einfachere Aufgaben zurückzuführen. Was wäre beispielsweise, wenn es sich statt um dreidimensionale um zweidimensionale Koordinaten handeln würde? Und wenn das nicht hilft, was wäre, wenn es sich um Punkte auf einer einzigen Linie handeln würde, sodass die Koordinaten nur einzelne Zahlen (etwa C++-Gleitkommazahlen doppelter Genauigkeit) sind? Nun ist die eigentliche Aufgabe, in einer Liste von Zahlen das Zahlenpaar zu finden, bei dem der Absolutbetrag der Differenz am kleinsten ist.

Sie könnten auch eine Reduktion des Problems vornehmen, indem Sie die Koordinaten zwar im dreidimensionalen Raum belassen, aber statt einer beliebig langen Folge nur drei Werte verwenden. Ein Algorithmus, der den kleinsten Abstand zwischen allen Koordinaten findet, ist nicht erforderlich. Es müssen nur noch die Koordinaten A und B, B und C sowie A und C miteinander verglichen werden.

Diese Reduktionen vereinfachen die Problemstellung auf verschiedene Weise. Im ersten Fall entfällt die Notwendigkeit, den Abstand zwischen Punkten im dreidimensionalen Raum zu berechnen. Vielleicht wissen wir noch nicht, wie das geht, aber bevor wir das herausfinden, machen wir dennoch Fortschritte in Richtung einer Lösung. Im Gegensatz dazu konzentriert sich die zweite Reduktion nahezu vollständig darauf, die Distanz zwischen dreidimensionalen Koordinaten zu berechnen, beseitigt aber das Problem, das Minimum einer beliebig langen Folge von Werten zu finden.

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278

Um die ursprüngliche Aufgabe zu lösen, sind natürlich letztlich die in beiden Reduktionen eingesetzten Fähigkeiten erforderlich. Dennoch erlaubt uns die Reduktion, an einem vereinfachten Problem zu arbeiten, sogar wenn es nicht gelingt, die Problemstellung in einzelne Schritte zu zerlegen. Faktisch handelt es sich hier um einen absichtlichen, wenn auch vorübergehenden, Kobayashi Maru. Uns ist klar, dass wir nicht am vollständigen Problem arbeiten, aber das vereinfachte Problem besitzt genug Gemeinsamkeiten mit dem vollständigen Problem, um es uns zu ermöglichen, Fortschritte auf dem Weg zur endgültigen Lösung zu machen. Es geschieht sehr oft, dass Programmierer feststellen, über alle erforderlichen Fähigkeiten zum Lösen einer Aufgabenstellung zu verfügen. Durch das Schreiben des Codes zum Lösen einzelner Aspekte der Problemstellung erkennen sie dann, wie sich die einzelnen Teile zu einem einheitlichen Ganzen kombinieren lassen.

Die Reduktion einer Problemstellung erlaubt es uns außerdem, festzustellen, wo genau die verbleibenden Schwierigkeiten liegen. Programmieranfänger sind oft auf die Hilfe erfahrener Programmierer angewiesen, jedoch kann dies für alle Beteiligten zu einem frustrierenden Erlebnis werden, wenn der sich mühende Programmierer nicht in der Lage ist, genau zu beschreiben, wobei er Hilfe braucht. Man sollte sich keinesfalls darauf beschränken zu sagen: »Hier ist mein Programm, das nicht funktioniert. Warum nicht?« Bei der Reduktion der Problemstellung kann die erforderliche Hilfe genau eingegrenzt werden: »Hier ist der Code, den ich geschrieben habe. Wie man sieht, weiß ich schon, wie der Abstand zwischen zwei dreidimensionalen Koordinaten berechnet wird, und ich weiß auch, wie ich prüfen kann, ob ein Abstand kürzer als ein anderer ist. Aber ich kann keine allgemeine Lösung finden, um das Koordinatenpaar mit minimalem Abstand zu berechnen.«

1.2.6 Analogien suchen

Für unsere Zwecke beschreibt eine *Analogie* eine Ähnlichkeit zwischen der vorliegenden Problemstellung und einem bereits gelösten Problem. Diese Übereinstimmung lässt sich zur Lösung des neuen Problems ausnutzen. Die Ähnlichkeit kann viele Formen annehmen. Manchmal bedeutet sie, dass es sich tatsächlich um ein und dieselbe Problemstellung handelt. Dieser Fall liegt beim Fuchs-Gans-Getreide-Rätsel und dem Quarrasi-Schloss vor.

Die meisten Analogien sind jedoch nicht derart unmittelbar. Manchmal betrifft die Ähnlichkeit auch nur Teile der Aufgabenstellung. Zwei Aufgabenstellungen zur Verarbeitung von Zahlen beispielsweise könnten sich grundlegend unterscheiden, mit der Ausnahme, dass beide mit Zahlen rechnen, die eine höhere Genauigkeit erfordern, als die verfügbaren Gleitkommadatentypen bieten. Beim Lösen des vollständigen Problems ist diese Analogie keine Hilfe, wenn Sie aber schon eine Möglichkeit gefunden haben, mit dem Problem der zu geringen

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278

Genauigkeit fertig zu werden, können Sie beide Problemstellungen auf die gleiche Weise lösen.

Zwar ist das Erkennen von Analogien die wichtigste Methode, mittels derer Sie Ihr Tempo und Können beim Problemlösen steigern werden, es ist jedoch gleichzeitig auch diejenige Fähigkeit, die am schwierigsten zu entwickeln ist. Der Grund für diese anfängliche Schwierigkeit ist, dass Sie nicht nach Analogien suchen können, wenn Sie nicht schon auf einen großen Schatz früherer Lösungen zurückgreifen können.

An dieser Stelle versuchen angehende Programmierer häufig, eine Abkürzung zu finden, indem sie nach Code suchen, der dem benötigten Code ähnelt, um diesen dann zu modifizieren. Dies zu tun, ist allerdings gleich aus mehreren Gründen ein Fehler. Erstens werden Sie den Code nicht hundertprozentig verstehen und verinnerlichen, wenn Sie ihn nicht vollständig selbst geschrieben haben. Vereinfacht ausgedrückt ist es sehr schwierig, ein Programm korrekt anzupassen, das Sie nicht komplett verstehen. Sie müssen den Code nicht selbst geschrieben haben, um ihn vollständig zu verstehen, aber wenn Sie den Code nicht selber hätten schreiben können, wird Ihr Verständnis notwendigerweise begrenzt bleiben. Zweitens ist jedes erfolgreiche Programm, das Sie schreiben, nicht nur die Lösung für ein aktuelles Problem, sondern potenzielle Quelle für Analogien zum Lösen künftiger Probleme. Je mehr Sie sich jetzt auf den Code anderer Programmierer verlassen, umso mehr werden Sie sich auch in Zukunft darauf verlassen müssen. Die sinnvolle bzw. nicht sinnvolle Wiederverwendung von Code kommt in Kapitel 7 noch ausführlich zur Sprache.

1.2.7 Experimentieren

Manchmal ist die beste Methode, um Fortschritte zu erzielen, einfach etwas auszuprobieren und das Resultat in Augenschein zu nehmen. Beachten Sie aber, dass Experimentieren und Raten zwei Paar Schuhe sind. Beim Raten geben Sie auf gut Glück etwas Code ein und hoffen, dass er funktioniert, ohne dabei einen sonderlich festen Glauben daran zu haben. Ein Experiment ist hingegen ein gesteuerter Vorgang. Sie stellen eine Hypothese darüber auf, was beim Ausführen eines bestimmten Codeabschnitts passieren wird, probieren es aus und überprüfen dann, ob Ihre Hypothese richtig war. Durch diese Beobachtungen erhalten Sie Informationen, die Ihnen dabei helfen, das ursprüngliche Problem zu lösen.

Experimentieren ist insbesondere beim Umgang mit APIs oder Klassenbibliotheken hilfreich. Angenommen, Sie schreiben ein Programm, das eine Bibliotheksklasse verwendet, die einen Vektor repräsentiert (in diesem Zusammenhang ein eindimensionales Array, das automatisch wächst, wenn ihm weitere Elemente hinzugefügt werden). Sie haben diese Vektorklasse aber vorher noch nie benutzt und sind sich nicht sicher, was geschieht, wenn ein Element des Vektors gelöscht wird. Anstatt sich der Lösung des Problems weiter zu nähern, obwohl Ihnen eine

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278

gewisse Unsicherheit im Kopf herumspukt, könnten Sie ein kleines separates Programm erstellen, nur um mit der Vektorklasse herumzuspielen und speziell die Situationen auszutesten, die Ihnen Kopfschmerzen bereiten. Wenn Sie einige Zeit mit dem Programm »Vektor-Demo« verbracht haben, könnte es Ihnen als Referenz dienen, wenn Sie die Klasse später erneut verwenden.

Andere Arten des Experimentierens sind dem Debuggen ähnlich. Nehmen wir an, ein bestimmtes Programm liefert Ausgaben nicht in der erwarteten Reihenfolge, sondern verkehrt herum. Wenn es sich beispielsweise um numerische Ausgaben handelt, sind die Zahlen an sich zwar korrekt, erscheinen jedoch in umgekehrter Reihenfolge. Wenn Ihnen nach der Überprüfung Ihres Codes nicht klar ist, warum dies geschieht, könnten Sie Ihren Code so abändern, dass er absichtlich Ausgaben in umgekehrter Reihenfolge erzeugt (etwa eine Schleife in der anderen Richtung durchlaufen). Die dadurch resultierenden – oder fehlenden – Änderungen in der Ausgabe könnten auf einen Fehler in Ihrem ursprünglichen Code hinweisen oder eine Lücke in Ihrem Verständnis offenbaren. Wie dem auch sei, Sie sind einer Lösung jedenfalls ein Stück näher gekommen.

1.2.8 Nicht entmutigen lassen

Bei der letzten Technik handelt es sich weniger um ein Verfahren als um einen Leitsatz: Lassen Sie sich nicht entmutigen. Wenn Sie frustriert sind, werden Sie nicht so klar denken wie sonst, nicht so effizient arbeiten wie sonst und alles wird länger dauern und Ihnen schwieriger erscheinen. Und was noch schlimmer ist: Frustration tendiert dazu, selbstverstärkend zu sein, und was als kleines Ärgernis beginnt, wird zu ausgemachtem Zorn.

Wenn ich Programmieranfängern diesen Ratschlag gebe, erwidern sie häufig, dass sie mir zwar im Prinzip zustimmen, sich aber außerstande sehen, das Maß ihrer Frustration zu steuern. Käme die Aufforderung an einen Programmierer, bei ausbleibendem Erfolg nicht frustriert zu sein, nicht der Bitte an einen kleinen Jungen gleich, nicht aufzuschreien, wenn er barfuß auf eine Heftzwecke tritt? Die Antwort lautet Nein. Wenn jemand auf eine Heftzwecke tritt, werden durch das zentrale Nervensystem augenblicklich starke Impulse gesendet, auf die tiefer liegende Ebenen des Gehirns ansprechen. Solange Sie nicht im Voraus wissen, dass Sie im Begriff sind, auf eine Heftzwecke zu treten, ist es unmöglich, rechtzeitig zu reagieren, um die automatische Rückmeldung des Gehirns zu unterdrücken. Dem kleinen Jungen sehen wir seinen Aufschrei daher nach.

Beim Programmierer sieht die Sache anders aus. Auch wenn ich nun Gefahr laufe, wie ein Motivationsguru zu klingen: Ein frustrierter Programmierer reagiert nicht auf einen von außen kommenden körperlichen Reiz. Ein frustrierter Programmierer ist auch nicht auf den Quellcode auf seinem Bildschirm wütend (obwohl manche ihrem Ärger manchmal in den Kommentaren darin Luft machen). In Wahrheit ist der frustrierte Programmierer auf sich selbst wütend.

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278

Die Quelle der Frustration ist zugleich deren Ziel, nämlich der Verstand des Programmierers.

Wenn Sie es sich erlauben, frustriert zu sein – ich verwende das Wort »erlauben« ganz bewusst –, geben Sie sich praktisch selbst einen Freibrief, um weiterhin zu scheitern. Nehmen wir an, dass Sie an einem hartnäckigen Problem arbeiten und spüren, wie allmählich die Frustration in Ihnen aufsteigt. Stunden später blicken Sie auf einen Nachmittag mit Zähneknirschen und vor Wut zerbrochenen Bleistiften zurück, und Sie sagen sich, dass Sie echte Fortschritte hätten erzielen können, wenn Sie dazu fähig gewesen wären, sich zu beruhigen. Tatsächlich haben Sie möglicherweise beschlossen, dass es einfacher ist, dem Ärger nachzugeben, als sich dem schwierigen Problem zu stellen.

Letzten Endes ist es Ihre Entscheidung, ob Sie der Frustration ausweichen können. Es gibt jedoch einige hilfreiche Überlegungen. Vergessen Sie vor allem die erste Regel nicht, nämlich stets über einen Plan zu verfügen, und dass es zwar das Hauptziel des Plans ist, Code zu schreiben, der die ursprüngliche Problemstellung löst, dieses aber nicht der einzige Schritt des Plans ist. Wenn Sie also über einen Plan verfügen und diesem folgen, dann machen Sie Fortschritte – daran müssen Sie schon glauben. Wenn Sie aber alle Schritte des ursprünglichen Plans abgearbeitet haben und sich noch immer nicht dafür bereit fühlen, mit dem Programmieren anzufangen, ist es an der Zeit, einen neuen Plan aufzustellen.

Sollten Sie außerdem feststellen, dass alles darauf hinausläuft, dass Sie frustriert werden, wenn Sie nicht mal Pause machen, dann sollten Sie diese Pause besser einlegen. Ein guter Trick ist es, stets mehr als eine Aufgabe bereitzuhalten, an der Sie arbeiten können. Wenn Sie bei einer bestimmten Aufgabe in der Klemme sind, können Sie sich dann einer anderen Aufgabe widmen. Beachten Sie hier, dass Sie nach erfolgreicher Aufgliederung einer Problemstellung in Teilaufgaben diese Vorgehensweise auch bei einer einzelnen Problemstellung einsetzen können. Dazu müssen Sie nur den Teil des Problems, bei dem Sie ins Stocken geraten sind, aus Ihren Gedanken verbannen und sich dann anderen Aufgaben zuwenden. Falls es keine sonstigen Aufgaben gibt, die Sie in Angriff nehmen könnten, erheben Sie sich von Ihrem Stuhl und machen Sie irgendetwas anderes, das Ihren Kreislauf in Schwung hält und Ihnen keine Kopfschmerzen bereitet. Unternehmen Sie einen Spaziergang, waschen Sie die Wäsche oder machen Sie Ihre Dehnübungen. (Falls Sie als Programmierer tätig sind und den ganzen Tag vor dem Computer sitzen, kann ich es nur wärmstens empfehlen, Dehnübungen zu machen!)

1.3 Übungen

Halten Sie sich vor Augen, dass es zum gründlichen Lernen unabdingbar ist, das Gelernte in die Praxis umzusetzen. Sie sollten daher möglichst viele der Übungen

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278

durcharbeiten. Im ersten Kapitel geht es natürlich noch nicht um Programmierung, aber dennoch möchte ich Sie ermuntern, einige der Übungen auszuprobieren. Stellen Sie sich diese Fragen als Fingerübungen eines Pianisten zum Aufwärmen vor, bevor das eigentliche Klavierkonzert beginnt.

1.1 Versuchen Sie, ein Sudoku mittleren Schwierigkeitsgrads zu lösen. Sie finden diese überall im Internet und vermutlich auch in Ihrer Lokalzeitung. Experimentieren Sie mit verschiedenen Strategien und notieren Sie sich die Resultate. Können Sie einen allgemeinen Plan zum Lösen von Sudokus aufstellen?

1.2 Betrachten Sie eine Schiebepuzzlevariante, bei der sich auf den Kacheln keine Zahlen befinden, sondern ein Bild. Inwieweit wird dadurch der Schwierigkeitsgrad erhöht und warum ist dem so?

1.3 Entwickeln Sie eine Strategie zum Lösen von Schiebepuzzles, die sich von meiner unterscheidet.

1.4 Suchen Sie nach antiquierten Varianten des Fuchs-Gans-Getreidesack-Rätsels und versuchen Sie, diese zu lösen. Viele der großartigen Rätsel wurden von Sam Loyd erfunden oder bekannt gemacht, suchen Sie daher am besten nach seinem Namen. Überlegen Sie sich außerdem, wie Sie das Rätsel vereinfachen könnten, sobald Sie die Lösung gefunden haben (oder aufgeben und diese nachlesen). Was müssten Sie ändern? Die Randbedingungen oder nur den Wortlaut des Rätsels?

1.5 Versuchen Sie, detaillierte Strategien für andere traditionelle Rätselspiele mit Papier und Bleistift zu entwickeln, wie beispielsweise Kreuzworträtsel. Wo sollte man anfangen? Was ist zu tun, wenn man nicht weiterkommt? Selbst so einfache Rätselspiele wie das in vielen Zeitungen vorhandene »Jumble« sind beim Nachdenken über Strategie nützlich.

© des Titels »Think Like a Programmer« (ISBN 978-3-8266-9278-9) 2013 by
Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg. Nähere Informationen unter: http://www.mitp.de/9278