

Inhaltsverzeichnis

Über den Autor	19
Danksagungen.....	19
Einführung.....	21
Teil I Lisp ist Power.....	31
1 Der Einstieg in Lisp.....	33
1.1 Lisp-Dialekte	33
1.1.1 Die Geschichte zweier Lisps	33
1.1.2 Neue, aufstrebende Lisps.....	34
1.1.3 Lisp-Dialekte für Scriptings.....	35
1.1.4 ANSI Common Lisp.....	35
1.2 Der Einstieg in CLISP	36
1.2.1 CLISP installieren	37
1.2.2 CLISP ausführen	37
1.3 Was Sie gelernt haben.....	38
2 Ein erstes Lisp-Programm erstellen.....	39
2.1 Das Errate meine Zahl-Spiel	39
2.2 Globale Variablen in Lisp definieren	40
2.2.1 Die Variablen *klein* und *gross* erstellen	40
2.2.2 Eine alternative Funktion zur Definition globaler Variablen.....	41
2.3 Grundlegende Lisp-Etikette	42
2.4 Globale Funktionen in Lisp definieren	42
2.4.1 Die errate-meine-zahl-Funktion definieren	43
2.4.2 Die Funktionen kleiner und groesser definieren	45
2.4.3 Die neu-starten-Funktion definieren	46
2.5 Lokale Variablen in Lisp definieren	46
2.6 Lokale Funktionen in Lisp definieren	47
2.7 Was Sie gelernt haben	48

3	Die Syntax von Lisp-Code erkunden	49
3.1	Syntax und Semantik	49
3.2	Die Bausteine der Lisp-Syntax	50
3.2.1	Symbole	51
3.2.2	Zahlen	51
3.2.3	Strings	52
3.3	Wie Lisp Code und Daten unterscheidet	53
3.3.1	Codemodus	53
3.3.2	Datenmodus	54
3.4	Listen in Lisp	54
3.4.1	Cons-Zellen	55
3.4.2	Listenfunktionen	56
3.4.3	Verschachtelte Listen	59
3.5	Was Sie gelernt haben	60
Teil II Lisp ist Symmetrie		63
4	Entscheidungen mit Bedingungen treffen	65
4.1	Die Symmetrie von nil und ()	65
4.1.1	Leer ist gleich falsch	65
4.1.2	Die vier Verkleidungen von ()	66
4.2	Die Bedingungsausdrücke: if und andere	68
4.2.1	Mit if eins nach dem anderen erledigen	68
4.2.2	Über if hinaus: die Alternativen when und unless	70
4.2.3	Der Befehl, der alles macht: cond	71
4.2.4	Verzweigungen mit case	73
4.3	Coole Tricks für Bedingungen	73
4.3.1	Die heimlichen Bedingungsbefehle and und or	73
4.3.2	Mit Funktionen arbeiten, die mehr als Wahrheitswerte zurückgeben	76
4.4	Dinge vergleichen: eq, equal und mehr	78
4.5	Was Sie gelernt haben	82
5	Eine Engine für ein Textspiel erstellen	83
5.1	Das Wizard's Adventure-Spiel	83
5.1.1	Unsere Spielwelt	84
5.1.2	Grundlegende Anforderungen	85
5.2	Die Szenerie mit einer Assoziationsliste beschreiben	85

5.3	Beschreibung des Schauplatzes	86
5.4	Beschreibung der Pfade	88
5.4.1	Wie Quasiquoting funktioniert	88
5.4.2	Mehrere Pfade auf einmal beschreiben	89
5.5	Objekte an einem bestimmten Schauplatz beschreiben	93
5.5.1	Sichtbare Objekte auflisten	93
5.5.2	Sichtbare Objekte beschreiben	94
5.6	Alles beschreiben	95
5.7	In unserer Welt herumwandern	96
5.8	Objekte aufheben	98
5.9	Unser Inventar prüfen	99
5.10	Was Sie gelernt haben	100
6	Mit der Welt interagieren: Lesen und drucken in Lisp	101
6.1	Text drucken und lesen	101
6.1.1	Auf dem Bildschirm drucken	102
6.1.2	Den Anwender begrüßen	103
6.1.3	Drucken und lesen – Die ersten Schritte	103
6.1.4	Dinge so lesen und drucken, wie Menschen es mögen	105
6.2	Die Symmetrie von Code und Daten in Lisp	107
6.3	Eine maßgefertigte Schnittstelle für unsere Game Engine	108
6.3.1	Eine maßgefertigte REPL einrichten	108
6.3.2	Eine maßgefertigte read-Funktion schreiben	109
6.3.3	Eine game-eval-Funktion schreiben	111
6.3.4	Eine game-print-Funktion schreiben	112
6.4	Unsere neue, aufgepeppte Spielschnittstelle ausprobieren	115
6.5	Die Gefahren von read und eval	117
6.6	Was Sie gelernt haben	118
6.5	Lambda: Eine Funktion, die so wichtig ist, dass sie ein eigenes Kapitel verdient	119
	Was lambda macht	119
	Warum lambda so wichtig ist	121
	Was Sie gelernt haben	121
7	Über grundlegende Listen hinaus	123
7.1	Exotische Listen	123
7.1.1	Punktlisten	124
7.1.2	Paare	125
7.1.3	Zirkuläre Listen	125

7.1.4	Assoziationslisten (Mappings)	127
7.2	Mit komplizierten Daten umgehen	128
7.2.1	Baumähnliche Daten visualisieren	128
7.2.2	Graphen visualisieren.	129
7.3	Einen Graphen erstellen	130
7.3.1	Die DOT-Daten generieren	131
7.3.2	Die DOT-Datei in eine Grafik umwandeln	135
7.3.3	Eine Grafik unseres Graphen erstellen.	139
7.4	Ungerichtete Graphen erstellen	140
7.5	Was Sie gelernt haben	143
8	Nicht das Wumpus, das unsere Väter kannten	145
8.1	Das Grand Theft Wumpus-Spiel	146
8.2	Die Kanten von Congestion City erstellen	150
8.2.1	Kanten zufällig generieren.	151
8.2.2	Schleifen mit der loop-Anweisung ausführen	152
8.2.3	Inseln (isolierte Teilgraphen) vermeiden	153
8.2.4	Die letzten Kanten für Congestion City erstellen.	155
8.3	Die Knoten von Congestion City erstellen	158
8.4	Ein neues Grand Theft Wumpus-Spiel initialisieren	161
8.5	Einen Plan unserer Stadt zeichnen	161
8.5.1	Eine Stadt auf der Basis partiellen Wissens zeichnen	162
8.5.2	Die Stadt erkunden.	165
8.6	Auf zur Wumpus-Jagd!	166
8.7	Was Sie gelernt haben	168
9	Fortgeschrittene Datentypen und generische Programmierung	169
9.1	Arrays	169
9.1.1	Mit Arrays arbeiten.	169
9.1.2	Mit einem generischen Setter arbeiten.	170
9.1.3	Arrays im Vergleich zu Listen	171
9.2	Hashtabellen	172
9.2.1	Mit Hashtabellen arbeiten	173
9.2.2	Mehrere Werte zurückgeben	174
9.2.3	Hashtabellen-Performance	175
9.2.4	Ein schnelleres Grand Theft Wumpus mit Hashtabellen	177
9.3	Common-Lisp-Strukturen	179
9.3.1	Mit Strukturen arbeiten	179
9.3.2	Wann man Strukturen einsetzen sollte	180

9.4	Daten generisch handhaben	182
9.4.1	Mit Sequenzen arbeiten	182
9.4.2	Eigene generische Funktionen mit Typ-Prädikaten erstellen	186
9.5	Das Orc Battle-Spiel	188
9.5.1	Globale Variablen für die Spieler und Monster	189
9.5.2	Hauptfunktionen des Spiels	189
9.5.3	Funktionen für das Spielermanagement	191
9.5.4	Hilfsfunktionen für Angriffe des Spielers	193
9.5.5	Funktionen für das Monstermanagement	194
9.5.6	Die Monster	195
9.5.7	Auf in den Kampf!	203
9.6	Was Sie gelernt haben	205
Teil III Lisp ist Codeschmiedekunst		207
10	Schleifen mit der loop-Anweisung ausführen	211
10.1	Das loop-Makro	211
10.1.1	Einige Schleifen-Tricks	212
10.1.2	Alles, was Sie schon immer über Schleifen wissen wollten	218
10.2	Evolution mit loop!	218
10.2.1	Pflanzen in unserer Welt wachsen lassen	219
10.2.2	Tiere erstellen	220
10.2.3	Einen Tag in unserer Welt simulieren	227
10.2.4	Unsere Welt darstellen	228
10.2.5	Eine Benutzerschnittstelle erstellen	229
10.2.6	Studieren wir einen Abschnitt der Evolution!	230
10.2.7	Der Ablauf der Evolution	235
10.3	Was Sie gelernt haben	236
11	Daten formatiert ausgeben	237
11.1	Anatomie der format-Funktion	237
11.1.1	Der Zielparameter	238
11.1.2	Der Kontrollstring-Parameter	238
11.1.3	Wertparameter	238
11.2	Kontrollsequenzen für die Ausgabe von Lisp-Werten	239
11.3	Steuersequenzen für die Formatierung von Zahlen	240

11.3.1	Steuersequenzen für die Formatierung von Ganzzahlen	240
11.3.2	Steuersequenzen zur Formatierung von Fließkommazahlen	241
11.4	Mehrere Output-Zeilen drucken	242
11.5	Output ausrichten	244
11.6	Listen mit Steuersequenzen durchlaufen	247
11.7	Ein verrückter Formatierungstrick zur Erstellung formatierter Tabellen	248
11.8	Angriff der Roboter!	249
11.9	Was Sie gelernt haben	250
12	Mit Streams arbeiten	253
12.1	Arten von Streams	253
12.1.1	Streams nach Ressourcentyp	253
12.1.2	Streams nach Richtung	254
12.2	Mit Dateien arbeiten	257
12.3	Mit Sockets arbeiten	259
12.3.1	Socket-Adressen	260
12.3.2	Socket-Verbindungen	261
12.3.3	Eine Message über ein Socket senden	261
12.3.4	Aufräumarbeiten	264
12.4	Stringstreams: Der Sonderfall	264
12.4.1	Streams an Funktionen übergeben	264
12.4.2	Mit langen Strings arbeiten	265
12.4.3	Lesen und Debuggen	265
12.5	Was Sie gelernt haben	266
13	Einen Webserver erstellen	267
13.1	Fehlerbehandlung in Common Lisp	267
13.1.1	Ein Problem anzeigen	267
13.1.2	Anwendungsspezifische Bedingungen erstellen	268
13.1.3	Abfangbedingungen	268
13.1.4	Ressourcen vor unerwarteten Problemen schützen	269
13.2	Einen Webserver von Grund auf schreiben	270
13.2.1	Wie ein Webserver funktioniert	270
13.2.2	Anforderungsparameter	272
13.2.3	Den Request-Header parsen	275
13.2.4	get-header mit einem Stringstream testen	277
13.2.5	Den Request-Body parsen	277

13.3	13.2.6 Unser großes Finale: Die serve-Funktion!	278
	Eine dynamische Website erstellen	279
	13.3.1 Den Request-Handler testen	280
	13.3.2 Die Website veröffentlichen	281
13.4	Was Sie gelernt haben.	281
<hr/>		
Teil IV Lisp ist Wissenschaft		301
<hr/>		
14	Mit funktionaler Programmierung in Lisp einen Gang zulegen	303
14.1	Was ist funktionale Programmierung?	303
14.2	Anatomie eines im funktionalen Stil geschriebenen Programms	306
14.3	Programmierung höherer Ordnung	309
	14.3.1 Codekomposition mit imperativem Code.	310
	14.3.2 Mit dem funktionalen Stil arbeiten.	310
	14.3.3 Rettung durch die Programmierung höherer Ordnung	311
14.4	Warum funktionale Programmierung verrückt ist.	311
14.5	Warum funktionale Programmierung fantastisch ist.	312
	14.5.1 Funktionale Programmierung verringert Fehler	312
	14.5.2 Funktionale Programme sind kompakter	313
	14.5.3 Funktionaler Code ist eleganter	313
14.6	Was Sie gelernt haben.	313
<hr/>		
15	Dice of Doom – Ein Spiel im funktionalen Stil	315
15.1	Die Regeln von Dice of Doom	315
15.2	Ein Probespiel von Dice of Doom	316
15.3	Implementierung von Dice of Doom, Version 1	317
	15.3.1 Einige globale Variablen definieren	318
	15.3.2 Das Spielfeld repräsentieren	318
	15.3.3 Die Regeln von Dice of Doom vom Rest des Spiels entkoppeln	320
	15.3.4 Einen Game Tree generieren	322
	15.3.5 Passzüge berechnen	323
	15.3.6 Angriffszüge berechnen	324
	15.3.7 Die Nachbarn ermitteln	325
	15.3.8 Angreifen	326
	15.3.9 Verstärkungen	327
	15.3.10 Unsere neue game-tree-Funktion ausprobieren	328
	15.3.11 Dice of Doom gegen einen anderen Menschen spielen	329

15.4	Einen intelligenten Computergegner erstellen	333
15.4.1	Der Minimax-Algorithmus.	334
15.4.2	Minimax in echten Code umsetzen.	334
15.4.3	Eine Spielschleife mit einem KI-Spieler erstellen	335
15.4.4	Unser erstes Mensch-gegen-Computer-Spiel spielen	336
15.5	Dice of Doom schneller machen	337
15.5.1	Closures.	338
15.5.2	Memoisierung.	340
15.5.3	Tail-Call-Optimierung	342
15.5.4	Ein Probespiel auf dem 3x3-Spielfeld	346
15.6	Was Sie gelernt haben	349
16	Die Magie der Lisp-Makros	351
16.1	Ein einfaches Lisp-Makro	352
16.1.1	Makroerweiterung	353
16.1.2	Wie Makros transformiert werden	354
16.1.3	Mit dem einfachen Makro arbeiten.	356
16.2	Komplexere Makros	356
16.2.1	Ein Makro zur Zerlegung von Listen	357
16.2.2	Wiederholte Ausführung in Makros vermeiden	358
16.2.3	Abfangen der Variablen vermeiden.	359
16.2.4	Ein Rekursionsmakro.	361
16.3	Makros: Gefahren und Alternativen.	363
16.4	Was Sie gelernt haben	365
17	Domain-spezifische Sprachen	367
17.1	Was ist eine Domain ³	367
17.2	SVG-Dateien schreiben	367
17.2.1	XML und HTML mit dem tag-Makro erstellen	369
17.2.2	SVG-spezifische Makros und Funktionen erstellen.	372
17.2.3	Ein komplizierteres SVG-Beispiel erstellen	374
17.3	Anwendungsspezifische Spielbefehle für das Wizard's Adventure-Spiel erstellen	376
17.3.1	Manuell neue Spielanweisungen erstellen.	377
17.3.2	Das fertige Wizards's Adventure-Spiel ausprobieren	383
17.4	Was Sie gelernt haben	385
18	Lazy Programming	387
18.1	Lazy Evaluation zu Lisp hinzufügen.	387
18.1.1	Die Anweisungen lazy und force erstellen	389

18.1.2	Eine Lazy-Lists-Bibliothek erstellen	391
18.1.3	Umwandlungen zwischen regulären Listen und Lazy Lists	392
18.1.4	Über Lazy Lists mappen und in Lazy Lists suchen	393
18.2	Dice of Doom, Version 2	395
18.3	Unsere KI für größere Spielfelder fit machen.	398
18.3.1	Den Game Tree beschneiden	398
18.3.2	Heuristiken anwenden.	400
18.3.3	Deutlich oder knapp gewinnen	401
18.3.4	Alpha Beta Pruning	404
18.4	Was Sie gelernt haben.	412
19	Eine grafische, webbasierte Version von Dice of Doom erstellen	413
19.1	Das Spielfeld im SVG-Format zeichnen	413
19.1.1	Einen Würfel darstellen	414
19.1.2	Ein einzelnes Feld darstellen.	417
19.1.3	Das Spielfeld darstellen	418
19.2	Die Webserver-Schnittstelle erstellen	420
19.2.1	Einen Web Request Handler schreiben	420
19.2.2	Einschränkungen unseres Spiel-Webservers	421
19.2.3	Ein neues Spiel initialisieren.	422
19.2.4	Einen Gewinner bekannt geben	422
19.2.5	Handhabung des menschlichen Spielers	422
19.2.6	Handhabung des Computerspielers	424
19.2.7	Das SVG-Spielfeld aus dem HTML-Code heraus zeichnen	424
19.3	Version 3 von Dice of Doom spielen	425
19.4	Was Sie gelernt haben.	427
20	Mehr Spaß mit Dice of Doom	429
20.1	Die Anzahl der Spieler erhöhen	429
20.2	Die Würfel werfen	430
20.2.1	Zufallsknoten erstellen	430
20.2.2	Die Würfel werfen	431
20.2.3	Die roll-dice-Funktion von der Game Engine aus aufrufen.	432
20.2.4	Die KI aktualisieren	433
20.3	Die Dice of Doom-Verstärkungsregeln verbessern.	435
20.4	Abschluss	437

A	Epilog	441
A.1	Kreuzer der funktionalen Gilde	452
	Lisp-Dialekt	452
	Übersicht	452
	Wie funktionale Programmierung Bugs bekämpft	452
	Beispiel	452
	Erklärung	452
	Schwäche	453
A.2	Nahkampfjäger der Makro-Gilde	454
	Lisp-Dialekt	454
	Übersicht	454
	Wie Makros Bugs bekämpfen	454
	Beispiel	454
	Erklärung	454
	Schwäche	454
A.3	Gepanzerte Jäger der Restart-Gilde	455
	Lisp-Dialekt	455
	Übersicht	455
	Wie Restarts Bugs bekämpfen	455
	Beispiel	456
	Erklärung	456
	Schwäche	456
A.4	Versorgungsschiff der generischen Setter-Gilde	458
	Lisp-Dialekt	458
	Übersicht	458
	Wie generische Setter Bugs bekämpfen	458
	Beispiel	458
	Erklärung	458
	Schwäche	458
A.5	Hot Rods der DSL-Gilde	461
	Lisp-Dialekt	461
	Übersicht	461
	Beispiel	461
	Erklärung	461
	Schwäche	461
A.6	Schlachtschiff der CLOS-Gilde	462
	Lisp-Dialekt	462
	Übersicht	462

Wie CLOS Bugs bekämpft	462
Beispiel 1: Methoden in Code einhüllen	462
Erklärung	462
Beispiel 2: Multiples Dispatching	463
Erklärung	463
Schwäche	464
A.7 Raketenkapseln der Continuation-Gilde	466
Lisp-Dialekt	466
Übersicht	466
Wie Continuations Bugs bekämpfen	466
Beispiel	466
Erklärung	467
Schwäche	467
A.8 Mikrojäger der Brevity-Gilde	472
Lisp-Dialekt	472
Übersicht	472
Wie Arc Bugs bekämpft	472
Beispiel	472
Erklärung	472
Schwäche	473
A.9 Formationsjäger der Multicore-Gilde	474
Lisp-Dialekt	474
Übersicht	474
Wie Software Transactional Memory Bugs bekämpft	474
Beispiel	474
Erklärung	474
Schwäche	475
A.10 Fregatte der Lazy-Gilde	475
Lisp-Dialekt	475
Übersicht	475
Wie Lazy Programming Bugs bekämpft	475
Beispiel	475
Erklärung	476
Schwäche	476
Stichwortverzeichnis	479