Titel: Land of Lisp. Lisp-Programmierung einfach lernen und originelle Spiele entwickeln
Autor: Conrad Barski
Jahr: 1. Auflage 2011
Verlag: mitp
Seitenzahl: 490 **Preis:** 34,95 EUR
ISBN: 978-3-8266-9163-8

1 Allgemeine Bemerkungen

Das Buch „Land of Lisp. Lisp-Programmierung einfach lernen und originelle Spiele entwickeln“ richtet sich an Leser mit und ohne Programmiererfahrung, die spielerisch einen der wichtigsten Lisp-Dialekte – Common Lisp – erlernen möchten.

2 Gliederung des Buches

Das Buch besteht aus vier Teilen:

1. Lisp ist Power (Kapitel 1 - 3)
2. Lisp ist Symmetrie (Kapitel 4 - 9)
3. Lisp ist Codeschmiedekunst (Kapitel 10 - 13)
4. Lisp ist Wissenschaft (Kapitel 14 - 20)

Kapitel 1: Der Einstieg in Lisp

Im ersten Kapitel werden die Lisp-Dialekte mit ihren Schwerpunkt im Einsatzgebiet vorgestellt:

- **Populärste Lisp-Dialekte:**
  - *ANSI Common Lisp*: wird im Buch verwendet.
  - *Scheme*
- ***Clojure***: verwendet Java-Bibliotheken, eignet sich für die Programmierung der Mehrkern-CPUs.
- ***Arc***: eine echte Lisp-Sprache, die sich noch im Entwicklungsstadium befindet.
- **Lisp-Dialekte für Scripting:**
  - *Emacs Lisp*
  - *Guile Scheme*
  - *Scritp-Fu Scheme*

Schließlich wird erläutert, wie *CLISP* installiert wird.

Kapitel 2: Ein erstes Lisp-Programm erstellen

Im zweiten Kapitel lernen wir, unser erstes Programm in *CLISP* in Form eines einfachen Spiels zu schreiben. Dabei wird erläutert, wie man in Lisp Definitionen machen kann. Es werden Definitionen von folgenden wichtigen Bestandteilen eines *Lisp*-Programms erläutert:

- globale vs. lokale Variablen
- globale Funktionen vs. lokale Funktionen

Kapitel 3: Die Syntax von Lisp-Code erkunden

Im dritten Kapitel wird auf die Syntax von *Lisp* genauer eingegangen. Dabei wird erläutert, welche Datentypen in Lisp verwendet werden:

- **Symbole**, die aus Buchstaben, Ziffern und Zeichen bestehen können
- **Zahlen**, die als Ganzzahlen und Fließkomazahlen verwendet werden können
- **Strings**, die durch Zeichen repräsentiert sind und in doppelte Anführungszeichen eingeschlossen werden.

Im weiteren Verlauf des Kapitels erfahren wir, wie sich in *Lisp* Code von Daten unterscheidet, nämlich durch den Code- bzw. Daten-Modus.

Schließlich wird das zentrale Konzept der *Listen* in Lisp vorgestellt. Dabei wird erläutert, wie eine Liste gebaut ist und welche elementaren Listenfunktionen es gibt:

- `cons`-Funktion
- `car`-Funktion
- `cdr`-Funktion

Auf diesen elementaren Listenfunktionen baut eine weitere sog. `list`-Funktion auf, mit deren Hilfe alle Cons-Zellen erstellt werden und die Liste in einem Zug aufgebaut wird. Schließlich erfahren wir, dass Listen in *Lisp* andere Listen enthalten können und der Autor erläutert, wie diese verschachtelten Listen realisiert werden.

Kapitel 4: Entscheidungen mit Bedingungen treffen

Im vierten Kapitel lernen wir, wie man in *Lisp* mit folgenden Befehlen Konditional-Ausdrücke schreiben kann:

- `if`-Anweisung
- `when`-Anweisung
- `unless`-Anweisung
- `cond`-Anweisung
- `case`-Befehl für Verzweigungen
- `and`- und `or`-Bedingungsbefehle

Es wird anschließend erläutert, wie man Funktionen schreiben kann, die mehr als nur Wahrheitswerte zurückgeben sollen. So zum Beispiel kann man mit dem Befehl `member` prüfen, ob ein Element in einer Liste enthalten ist.

Weiterhin wird ein Befehl `find-if` vorgestellt, der bestimmte Werte abruft und der als boolischer Wert in einer Bedingung dienen kann.

Schließlich werden Vergleichsoperatoren vorgestellt:

- `eq`: für den Vergleich von Symbolen
- `eql`: für den Vergleich für Zahlen und Zeichen
- `equal`: für den Vergleich von Strings und Zeichen
- `equalp`: für den Vergleich von Strings mit Groß-/Kleinschreibung und Ganz-Zahlen mit Fließkommazahlen

Kapitel 5: Eine Engine für ein Textspiel erstellen

Im fünften Kapitel beschreibt der Autor die Handhabung von Texteingabe und -ausgabe in *Lisp*. Dabei lernen wir, wie man Texte anhand der Programmierung eines einfachen *Wizard's Adventure*-Spiels manipulieren kann. Wir erfahren, wie man Assoziationslisten erstellen kann.

Kapitel 6: Mit der Welt interagieren: Lesen und drucken in Lisp

Im sechsten Kapitel lernen wir, die Texte direkt auf dem Bildschirm zu drucken sowie vom Benutzer eingegebenen Text einzulesen. Dafür verwendet man folgende Lisp-Befehle auch für die Konsolenein- und ausgabe:

- `print`: gibt jeden Wert in einer neuen Zeile mit abschließenden Leerzeichen aus.
- `prin1`: gibt Werte ohne Zeilenumbruch aus.
- `princ`: Lisp-Daten werden vor der Ausgabe in der Hinsicht aufbereitet, dass sie für Menschen lesbarer werden, nämlich ohne Anführungszeichen.
- `read`: zum Einlesen der Werte.

Anschließend werden beide Funktionen am bereits angefangenen Adventure-Spiel eingesetzt.

Schließlich werden Gefahren von `read`- Funktion beschrieben. So kann der `read`-Befehl Bestandteil eines *Reader*-Makros sein, und beim Lesen der Daten einen schädlichen Computercode ausführen.

Schließlich wird noch der `lambda`-Funktion ein eigenes Unterkapitel gewidmet. Es wird erläutert, wie man mit `lambda` eine anonyme Funktion erstellen kann und wie eine Funktion als Parameter an eine andere Funktion übergeben wird (was als *Programmierung höherer Ordnung* bezeichnet wird).

Kapitel 7: Über grundlegende Listen hinaus

Im siebten Kapitel erfahren wir, wie man in *Lisp* verschiedene Arten von Listen erstellen kann. Die einfachste Liste wird mithilfe von `cons`-Zellen erstellt, deren letzter Slot immer mit `nil` endet. Mit mehreren `cons`-Zellen kann man beliebig lange Listen erstellen:

- Punktlisten
- Paare
- Zirkuläre Listen
- Assoziationslisten

Im weiteren Verlauf des Kapitels lernen wir Techniken kennen, um komplizierte Daten bequem zu visualisieren wie z.B. baumähnlich strukturierte Daten mithilfe von mathematischen Graphen.

Kapitel 8: Nicht das Wumpus, das unsere Väter kannten

Im achten Kapitel wird ein anspruchsvolleres Spiel entwickelt, bei dessen Programmierung die vorher erwähnten Graph-Utilities verwendet werden. Ausserdem wird der Einsatz von folgenden Funktionen beschrieben:

- Die `loop`-Funktion, mit deren Hilfe verschieden Datentypen durchlaufen werden können.
- Die `set-diffrenece`-Funktion, mit deren Hilfe man erfahren kann, welche Elemente in einer, aber nicht in einer anderen Liste enthalten sind.
- Die `remove-duplicates`-Funktion, mit deren Hilfe die doppelten Elemente aus einer Liste entfernt werden können.

Kapitel 9: Fortgeschrittene Datentypen und generische Programmierung

Im neunten Kapitel werden folgende Datentypen behandelt, die ihren Einsatz im Spiel aus dem vorigen Kapitel (*Grand Theft Wumpus*) finden:

- Arrays
- Hashtabellen

Im weiteren Verlauf des Kapitels stellt der Autor noch einen Datentyp vor: *Struktur*, mit deren Hilfe Daten in Code repräsentiert werden können. Die *Lisp*-Strukturen können Objekte mit ihren Eigenschaften wie in objektorientierten Sprachen repräsentieren. In *Lisp* werden Strukturen mittels des `defstruct`-Befehls definiert. Dabei gibt uns der Autor Tipps, wann man Strukturen einsetzen soll.

Im weiteren Verlauf des Kapitels erläutert der Autor, wie *Lisp* Daten generisch handhabt. In diesem Zusammenhang werden folgende Sequenz-Funktionen vorgestellt, die generisch mit den drei Hauptarten von Sequenzobjekten in *Lisp* arbeiten, d.h. mit Listen, Arrays und Strings:

- `length`: Abfrage der Länge eines beliebigen Sequenztyps.
- `find-if`: Suchfunktion zur Ermittlung des ersten Wertes, der ein Prädikat erfüllt.
- `count`: Ermittlung des Vorkommens eines bestimmten Objektes in einer Sequenz.
- `position`: Ermittlung und Rückgabe der Position eines Elements.
- `some` und `every`: Ermittlung von allen oder einigen Objekten in einer Sequenz mit dem bestimmten Prädikat.
- `reduce`: Reduzierung der Sequenz auf ein Ergebnis bzw. Destillieren eines einzigen Ergebnisses aus einer Sequenz.
- `map`: der Sequenztyp wird als zusätzliches Argument an die `map`-Funktion übergeben und soll durch diese Mapping-Operation zurückgegeben werden.

Weiter im Kapitel wird erläutert, wie man eigene generische Funktionen mit Typ-Prädikaten erstellen kann.

Schließlich können wir ein neues Spiel „Orc Battle“ mitkodieren, wobei jeder Codeblock auf viele Haupt-Funktionen sowie Hilfsfunktionen analysiert und erklärt wird.

Kapitel 10: Schleifen mit der Loop-Anweisung ausführen

Im zehnten Kapitel lernen wir, wie man in *Lisp* Schleifen mit loop-Anweisungen erstellen kann. Dazu kann man ein `loop`-Makro anwenden, mit dessen Hilfe jede Art von Schleifen realisiert werden kann.

Am Ende des Kapitels wird die Kodierung eines Spiels vorgestellt, in dem man Schleifen vermehrt einsetzen kann, nämlich um eine wachsende Pflanzenwelt zu realisieren bzw. sich bewegende Tiere oder deren Reproduktion zu kodieren. Es wird hierzu auch eine Benutzerschnittstelle erstellt.

Kapitel 11: Daten formatiert ausgeben

Im elften Kapitel stellt der Autor verschiedene Arten der Datenformatierung in *Lisp* vor. Dabei geht er auf die Formatierung verschiedener Datentypen wie z.B. von Zeichen oder Zahlen. Außerdem wird erläutert, wie man Tabellen in *Lisp* formatiert.

Kapitel 12: Mit Streams arbeiten

Im zwölften Kapitel wird die Handhabung von Streams seitens *Lisp* erläutert. Streams spielen in *Lisp* eine wichtige Rolle bei verschiedenen Arten von Interaktionen. Dabei werden Streams je nach der Ressource und der Richtung in verschiedene Typen unterteilt:

- **Typen von Streams je nach Ressource:**
  - Konsolenstreams: Eingabe und Ausgabe von Daten über die Konsole (REPL).

  - Filestreams (Dateistreams): Schreiben der Daten in Dateien auf einem externen Speicher bzw. Einlesen aus ihnen.
  - Socketstreams: Kommunikation mit anderen Computern in einem Netzwerk.
  - Stringstreams: Herauslesen des Textes aus einem *Lisp*-String bzw. dessen Speicherung in einen String.
- **Typen von Streams je nach Richtung:**
  - Outputstreams: Daten werden in REPL ausgegeben, in eine Datei geschrieben oder über einen Socket gesendet.
  - Inputsreams: Daten werden in ein Programm eingelesen.

Es wird außerdem vorgestellt, wie man mithilfe von der `with-open-file`-Funktion Daten in Dateien schreiben bzw. aus Dateien einlesen kann. Dabei wird diese Funktion mit zusätzlichen Funktionen für den Output (`:direction :output`) bzw. für den Input (`:direction :input`) versehen, um die Richtung der fließenden Daten anzugeben.

Im weiteren Verlauf des Kapitels wird die Erstellung von Socket-Verbindungen erläutert.

Schließlich erfahren wir, wie in Lisp Stringstreams behandelt werden. Dabei wird erläutert, wie Stringstreams als Funktionen verwendet werden und andere Streams als Parameter haben können. Somit werden solche Stringstreams für die Suche von Fehlerquellen (Debuggen) eingesetzt.

Kapitel 13: Einen Webserver erstellen

Im dreizehnten Kapitel stellt der Autor Techniken zur Erstellung eines Webservers von Grund auf.

Zunächst geht der Autor sehr ausführlich auf das Thema der Fehlerbehandlung unter *Lisp* ein:

- Anzeigen der Fehler;
- Abfangbedingungen für die Fehler nennen;
- Ressourcen vor unerwarteten Problemen schützen

Anschließend erklärt der Autor, wie ein Webserver überhaupt funktioniert, welche Anforderungsparameter es für Webformulare gibt (z.B. GET-Anforderungen). Wir erfahren, wie man den Request-Header bzw. Request-Body parsen kann. Anschließend wird eine `serve`-Funktion vorgestellt.

Schließlich stellt der Autor Techniken zur Erstellung einer dynamischen Website unter *Lisp* vor und demonstriert wie man die Website veröffentlicht.

Kapitel 14: Mit funktionaler Programmierung in Lisp einen Gang zulegen

Im vierzehnten Kapitel geht der Autor ausführlich auf funktionales Programmieren in *Lisp* ein. Zunächst wird erklärt, was funktionale Programmierung ausmacht. Anschließend wird die Anatomie eines im funktionalen Stil geschriebenen Programms unter die Lupe genommen.

Im späteren Verlauf des Kapitels wird das Konzept der Programmierung höherer Ordnung vorgestellt.

Schließlich werden positive Seiten der funktionalen Programmierung aufgezählt und mit Argumenten belegt.

Kapitel 15: Dice of Doom - Ein Spiel im funktionalen Stil

Im fünfzehnten Kapitel wird an einem neuen Spiel „Dice of Doom“ die Programmierung im funktionalen Stil demonstriert und ausführlich erklärt. Dabei wird eine Optimierung des Spiels mittels sog. *Closures* vorgenommen. *Closures* in *Lisp* stellen hier zusätzliche Daten aus der Außenwelt dar, die bei der Erstellung von Lambda-Funktionen erfasst werden. Sie werden zur Zwischenspeicherung kleiner Datenmengen zwischen den Aufrufen einer Funktion verwendet.

Als weitere Optimierung eines Programms im funktionalen Still dient die sog. *Memoisierung*, die mit *Closures* arbeitet. Mit ihrer Hilfe wird eine erneute Berechnung eines zuvor berechneten Ergebnisses einer Funktion eingespart.

Als dritte Optimierungstechnik für funktionale Programme stellt der Autor die sog. *Tail-Call-Optimierung* vor.

Kapitel 16: Der Magier der Lisp-Makros

Im sechzehnten Kapitel wird die Funktionalität der *Lisp*-Makros näher erläutert, mit denen man einen Code schreiben kann, der wiederum einen Code generiert. Somit wird eine Grundlage für die Erstellung von eigenen Programmiersprachen gelegt. Wir lernen dabei, wie man Makros transformieren aber auch wie man die wiederholte Ausführung in Makros vermeiden kann. Außerdem wird erläutert, wie man ein Rekursionsmakro erstellen kann.

Schließlich werden Gefahren und Alternativen von Makros aufgezählt.

Kapitel 17: Domain-spezifische Sprachen

Im siebzehnten Kapitel wird eine fortgeschrittene Makroprogrammiertechnik – die *DSL*-Programmierung – vorgestellt, die es ermöglicht Domain-spezifische Lösungen für ein bestimmtes Programm zu realisieren. Dabei wird die Art der DSL-Programmierung exemplarisch an zwei Fällen vorgeführt: SVG-Dateien schreiben und das Spiel „Wizards Adventure“ uneingeschränkt spielbar machen.

Für das Programm zur Erstellung von SVGs werden Makro-Funktionen geschrieben, die es ermöglichen, das XML-Format einer SVG-Datei durch das tag-Makro zu automatisieren. Beim Spiel „Wizards Adventure“ wird ein Makro erstellt, welches die manuelle Erstellung der neuen Spielanweisungen ermöglichen soll.

Kapitel 18: Lazy Programming

Im achtzehnten Kapitel wird das sog. Lazy Programming, vorgestellt, dessen Konzept auf Lazy Evaluation basiert, d.h es werden nicht alle Teile des Codes ausgewertet, um so die Ausführung des Programms zu beschleunigen. Es wird erläutert, wie man mit den zwei grundlegenden Befehlen `lazy` und `force` in *Lisp* diese *Lazy Evaluation* ermöglicht.

Im weiteren Verlauf des Kapitels wird demonstriert, wie man mittels des `lazy-cons`-Befehles eine *Lazy-Lists*-Bibliothek erstellen, sowie *Lazy-Lists* mappen und suchen kann. Ferner wird erläutert, wie man Umwandlungen zwischen regulären Listen und *Lazy Lists* realisieren kann.

Schließlich wendet der Autor alle neuen Erkenntnisse am Code des Spiels „Dice of Doom“ an und schafft dabei eine zweite Version dieses Spiels.

Kapitel 19: Eine grafische, webbasierte Version von Dice of Doom erstellen

Im neunzehnten Kapitel wird das Spiel „Dice of Doom“ um eine grafische, webbasierte Version erweitert. Dazu werden alle Darstellungen von grafischen Oberflächen mittels des neuen SVG-Makros codiert: Spielfeld, Würfel.

Anschließend wird für das Würfel-Spiel eine Webserver-Schnittstelle erstellt. Es wird dabei erläutert, wie man ein neues Spiel initialisieren, sowie die Funktionalität der Bekanntgabe des Gewinners implementieren kann. Es werden Funktionalitäten hinzugefügt, die eine entsprechend angepasste Handhabung für ein Spiel mit menschlichen bzw. Computerspielers realisieren.

Schließlich erfahren wir, wie man das SVG-Spielfeld aus dem HTML-Code heraus zeichnen kann und damit die dritte Version von „Dice of Doom“ abspielen kann.

2.1 Kapitel 20: Mehr Spaß mit Dice of Doom

Im zwanzigsten Kapitel wird die letzte Version von „Dice of Doom“ mit weiteren Funktionserweiterungen geschrieben. Dabei wird zunächst die Anzahl der Spieler erhöht, die Funktion zum Würfel-Werfen mit Zufallsknoten versehen usw.

2.2 Epilog

Im letzten Teil des Buches finden wir den Epilog mit der Aufzählung von Stärken und Schwächen von ausgewählten Lisp-Dialekten, die mit Comic-Zeichnungen begleitet werden:

- Common Lisp
- Scheme
- Arc Lisp
- Clojure Lisp

3 Kritik

Für den inhaltlichen Teil des vorliegenden Buches kann man nur eine positive Kritik äußern. Methodisch sehr gut aufbereitet, führt das Buch von niedrigerem Level der Programmierung in Lisp zu höherer Programmierung (wie z.B. einen Webserver aufsetzen uvm.) sehr gelungen durch.

Als störend wurde bei diesem Buch empfunden, dass die Zuordnungsnummern der Erklärungen zu den Listings teilweise nicht stimmten oder gar fehlten. Auch bei den Listings selbst fehlen Ziffern, die dann später im Text mit Nummern erläutert werden. Hier ist unbedingt die Lektoratsabteilung nochmal gefragt!

3.1 Tippfehler

Seite 19 „Danksagungen“ zweiter Absatz, zweiter Satz *dem Hauptlektor für den größten Teil dieses Projekt.*
muss geändert werden in:
Projektes

Seite 56 erster Absatz, dritter Satz *... beispielsweise ein Anrufkette ...*
muss geändert werden in:
eine

Seite 56 "Die cons-Funktion“ zweiter Absatz von unten, zweiter Satz *Es sorgt dafür sorgt, dass ...*
Das zweite *sorgt* muss entfernt werden.

Seite 59 „Die list-Funktion“ zweiter Absatz, zweiter Code-Beispiel *(LIST 'SCHWEIN 'RIND' 'HUHN*
muss geändert werden in:
'RIND (ohne Apostroph hinten)

Seite 63 „Kapitel 4“ Die Überschrift mit Seitenangabe *Der Einstieg in Lisp, 33*
muss geändert werden in:
Entscheidungen mit Bedingungen treffen, 65

Seite 99 zweiter Absatz, zweiter Satz nicht-verständlich bzw. unvollständig *bedeutet dies, dass ein Objekt mehrfach mit unterschiedlichen Orten in* `*objekt-standorte*` *enthalten sein.*

Seite 114 erster Absatz, dritter Satz *Dies ist eine der viele print-Varianten ...*
muss geändert werden in:
vielen

Seite 151 „Kanten zufällig generieren“ :

Zu der zweiten Funktion *`(defun edge-pair (a b)...)`* fehlt die Erläuterung.

Der Nummern-Verweis 3 bei der Erläuterung für den `loop`-Befehl in:
(*apply #'append (loop repeat *edge-num*...)*) muss auf Nummer 4 ersetzt werden.

Seite 165 oberer Satz *Jetzt können können wir ...*
Das zweite *können* muss entfernt werden.

Seite 166 *Dann ruft sie **draw-known-city** erneut auf, da wir jetzt einen neuen Schauplatz kennen*:
Der Nummern-Verweis 5 bei dieser Erläuterung muss eingefügt werden.

Seite 166 letzter Absatz von Kapitel 8.5.2, letzter Satz ist nicht-verständlich bzw. es fehlt ein Verb: *Wenn sich an dem Ort schließlich eine Glowworm-Gang, auf die wir noch nicht gestoßen sind, wird **handle-new-place** rekursiv aufrufen und wir werden an einen zufällig ausgewählten neuen Standort versetzt.*

Seite 176 letzter Absatz, erster Satz *Es gibt noch einen letzten Grund, warum Hashtabellen sind nicht immer die beste Lösung sind ...*
Das erste *sind* muss entfernt werden.

Seiten 190-191, 223 :

Bei den Listing-Beispielen müssen alle Ziffern überprüft und korrigiert werden (sie fehlen an manchen Stellen oder werden falsch der Stelle im Listing zugeordnet).

Seite 194 zweiter Absatz, letzter Satz *... und wir geben das ausgewählte Monster als Ergebnis zurück (2)*
muss geändert werden in:
(6)

Seite 201 erster Absatz, erster Satz *Er kann Ihnen auch auch Schleim ins Gesicht spritzen.*
Das zweite *auch* muss entfernt werden.

Seite 211 erster Absatz, letzter Satz *In diesem Kapitel wird der `loop`-Befehl behandelt, im nächsten Kapitel der und `format`-Befehl.*
und muss entfernt werden.

Seite 215 dritter Absatz, erster Satz *Manchmal werden Sie jedoch das kartesische Produkt mehrerer Wertebereich erzeugen wollen.*
muss geändert werden in:
Wertebereiche

Seite 223 vorletzter Absatz, erster Satz *Um die neue X-Koordinate zur berechnen...*
muss geändert werden in:
zu

Seite 225 zweiter Absatz, erster Satz *Die Energie des Tieres wir um den Energiewert gesteigert ...*
muss geändert werden in:
wird

Seite 228 erster Absatz, erster Satz *Schließlich rufen wir die add-plants-Funktion auf (1)...*
muss geändert werden in:
(3)

Seite 255 letzter Absatz, erster Satz *Mit der `input-stream-p`-Funktion können Sie prüfen, ob ein Outputstream gültig ist.*
muss geändert werden in:
Inputstream

Seite 279 erster Absatz, dritter Satz *(Port 80 wir normalerweise für eine ...)*
muss geändert werden in:
wird

Seite 279 zweiter Absatz, vorletzter Satz *Aus Kapitel wissen Sie 12, dass ...*
muss geändert werden in:
Aus Kapitel 12 wissen Sie, dass ...

Seite 307 letzer Absatz, letzer Satz - 308, erster Satz *Sie besteht nur aus einer Lisp-Liste, die in der globalen Variablen *database* gespeichert ist.*
muss geändert werden in:
Variable

Seite 310 14.3.1, zweiter Absatz, dritter Satz *Dies ähnelt Code, den Sie ...*
muss geändert werden in:
Dies ähnelt dem Code, den Sie ...

Seite 318 erster Absatz, erster Satz *Zuerst definieren wir ein paar globale Variable für ...*
muss geändert werden in:
Variablen

Seite 324 dritter Absatz, letzter Satz *Der Spielerwechsel erfolgt an dieser Stelle auf so aufwendig, damit ...*
auf muss entfernt werden.

Seite 333 vorletzter Absatz, letzter Satz *Um die Bewertung in diesen Fällen zu ermitteln, müssen wir einbeziehen, die der Gegner reagieren wird.*
die muss durch *wie* ersetzt werden.

Seite 356 16.1.3 vorletzter Absatz, zweiter Satz *Das T am Ende (1) teilt uns mit, dass ...*

muss geändert werden in:
(2)

Seite 372 17.2.2 Listing muss mit Ziffern versehen werden.

Seite 373 vorletzter Absatz, letzter Satz *Der Radius wird mit dem Attribut* `r` *gesetzt (1) und der Stil mit unserer* `svg-style`*-Funktion (2).*
muss geändert werden in:
(1) -> (3), (2) -> (4)

Seite 376 erster Absatz,letzter Satz *Um das Ganze etwas bunter zu machen, wählen wir zusätzlich die Farbe der Kanten zufällig aus (1)*
muss geändert werden in:
(5)

Seite 382 erster Absatz, erster Satz *... für einen Befehl definieren (1)*
muss geändert werden in:
(2)

Seite 382 dritter Absatz, vorletzter Satz *... geben wir ... mit dem Namen des gegenwärtigen Befehls aus (2).*
muss geändert werden in:
(6)

Seite 382 dritter Absatz, letzter Satz *Schließlich fügen wir ... in die Liste der zufälligen Befehle unserer angepassten* `game-repl` *(3).*
muss geändert werden in:
(7)

Seite 389 vorletzter Absatz, letzter Satz *... da die Meldung „Ich addiere jetzt“ noch nicht anzeigt wurde.*
muss geändert werden in:
angezeigt

Seite 390 zweiter Absatz, erster Satz *... mit den erstellten* `gensym`*-Namen zwei lokale Variablen deklariert (3).*
muss geändert werden in:
(2)

Seite 391 18.1.2 erster Absatz, erster Satz *... die an die entsprechend Implementierung ....*
muss geändert werden in:
entsprechende

Seite 394 erster Absatz, dritter Satz *Diese bedeutet, dass sie nicht ....*
muss geändert werden in:
Dies

Seite 406 dritter Absatz von unten, erster Satz *Diese bedeutet, dass sich der ....*
muss geändert werden in:
Dies

Seite 419 vorletzter Absatz, zweiter Satz *... verwendet, das wir hier erstellen (2).*
muss geändert werden in:
(6)

Seite 421 19.2.1 vorletzter Satz *... und übergeben die Kontrolle an web-handle-computer (1), um ...*
muss geändert werden in:
(9)

Seite 423 letzter Absatz, erster Satz *Ist die gegenwärtig ausgewählte Position dieselbe wie *from-title*-Variable, ist ein Feld zweimal ausgewählt werden.*
muss geändert werden in:
worden

Seite 458 letzter Absatz, letzter Satz *Dank dieser Funktion können die meist Arten von Datenstrukturen ....*
muss geändert werden in:
meisten

4 Fazit:

Das Buch „Land of Lisp. Lisp-Programmierung einfach lernen und originelle Spiele entwickeln“ ist eine ungewöhnlich gelungene Übersetzung des englischen Originals von Conrad Barski durch Reinhard Engel.

Mit diesem Buch führt uns der Autor durch die Welt der Lisp-Programmiersprache von den Grundlagen bis zur fortgeschrittenen Programmierung. Dabei lädt er uns ein, mit ihm gemeinsam zunächst einfache Spiele zu programmieren, die später aber zu immer mehr komplexeren Adventure-Spielen ausgebaut werden.

Das Buch eignet sich besonders für Programmier-Anfänger, da deren Aufmerksamkeit und der Spaß am Lesen sowie am Ausprobieren des Lisp-Codes nicht nur durch lustige Comic-Zeichnungen stets erhalten bleibt, die das ganze Buch durchziehen, sondern auch durch die hervorragende inhaltliche Progression, die den Leser immer dort abholt, wo er sich gerade befindet.

So wie der Autor davon überzeugt ist, dass der Leser sich nach dieser Einführung in einen Programmierer verwandeln wird, dessen Verständnis für nicht-Lisp-Programmiersprachen eine grundlegende Veränderung erfährt, so bin ich ebenfalls davon überzeugt worden, dass es ihm dies zumindest in Bezug auf mich vollständig gelungen ist: Ich mag jetzt LISP!