Dokumentation zu dem EMACS-Major-Mode für Phutball

Robert Höhndorf

22. August 2002

Zusammenfassung

EMACS ist ein von erfahrenen Programmierern vielgeschätzter Editor. Besonders an EMACS ist seine hervorragende Erweiterbarkeit unter Verwendung von ELisp, einem Lisp-Dialekt. Doch lässt sich EMACS auch für eine Vielzahl anderer Anwendungen verwenden, unter anderem existieren bereits Webbrowser, Mail- und Newsclients, FTP-Clients und vieles mehr für EMACS. Hier wird die Implementierung einer Erweiterung für ein Zwei-Personen-Spiel beschrieben, mit dem sich über ein Netzwerk Phutball spielen lässt. Die Besonderheiten der Verwendung einer Sprache wie Lisp sowie der gewählten Plattform EMACS werden erörtert. Ausserdem wird die Vorgehensweise bei der Implementierung und die eigentliche Implementierung der Erweiterung beschrieben.

Inhaltsverzeichnis

1 Einführung

Hier werden die Aufgabenstellung sowie allgemeine Konzepte für die Umsetzung beschrieben.

1.1 Aufgabenstellung

Die Aufgabe bestand in der Implementierung eines Frontends für das Zwei-Personen-Spiel Phutball, für welches im Sommersemester 2002 an der Universität Leipzig im Rahmen eines Wettbewerbes Spielprogramme implementiert wurden. Da die Implementierung dieses Frontends im Rahmen einer praktischen Arbeit für die Vorlesung "Funktioniale Programmierung 1" stattfinden sollte, war die Verwendung einer Funktionalen Programmiersprache vorgeschrieben.

Im Rahmen des Wettbewerbes wurde ein Spielserver entwickelt, über welchen zwei Spieler gegeneinander ein Phutballspiel austragen können. Der Server verwendet TCP/IP. Zur Lösung der Problemstellung galt es also, nicht nur ein Frontend zur Verfügung zu stellen, sondern auch über ein Netzwerk mit dem Spielserver zu kommunizieren.

1.2 Das Zwei-Personen-Spiel Phutball

Phutball - "Philosophen Fussball" - ist ein Zwei-Personen-Spiel, welches von John H. Conway erfunden wurde. Das Spielfeld ist eine Matrix, welche meist 15 Felder breit und 19 Felder hoch ist, aber auch beliebige andere Ausmasse annehmen kann. Auf dem Feld befinden sich genau ein Ball sowie beliebig viele Spielsteine ("Männer"). Ziel des Spieles ist es, den Ball regelgemäss auf oder über eine Torlinie zu befördern. Diese Torlinie ist jeweils die erste beziehungsweise die letzte Zeile der Matrix.

Um dieses Ziel zu erreichen sind genau zwei Züge erlaubt: Man kann einen Spielstein auf ein freies Matrix-Feld setzen oder mit dem Ball adjazente Spielsteine überspringen. Nach einem Sprung werden die übersprungenen Steine entfernt und der Sprung kann fortgesetzt werden. Nachdem ein Spielzug ausgeführt wurde, zieht der Gegenspieler. Es wird keine Unterscheidung zwischen Spielsteinen der beiden Gegenspieler getroffen. Auch sind weitere Spielzüge wie zum Beispiel "Passen" nicht erlaubt.

Das Spiel Phutball gehört in die Klasse der schweren Spiele, das heisst, es ist nicht davon auszugehen, dass in absehbarer Zeit eine vollständige Theorie

für das Spiel gefunden wird. Für viele Spiele ist bekannt, welche Komplexität das Problem, den Ausgang eines Spieles zu ermitteln, besitzt. Viele interessante Spiele liegen dabei in Komplexitätsklassen von PSPACE oder gar EXPTIME (Schach, Go). Ein ähnliches Resultat ist für Phutball nicht bekannt.

Demaine, Demaine und Eppstein gelang es aber zu zeigen, dass bereits das Ermitteln, ob in einer bestimmten Spielsituation ein Sieg-Sprung möglich ist, ein NP-vollständiges Problem ist. [4, Seiten 21ff]

1.3 EMACS

Laut seinem Handbuch ist Emacs ein "erweiterbarer, anpassbarer, selbstdokumentierender Echtzeit-Display-Editor".

Emacs ist also ein Texteditor und mehr. Sein Kern besteht aus einem Interpreter für Emacs Lisp (kurz: Elisp), einem Lisp-Dialekt mit Erweiterungen, um das Editieren von Texten zu ermöglichen.

Emacs besteht aus fünf Komponenten: Ein- und Ausgabeprimitiven, dem Interpreter, dem Command Dispatcher, dem Library System und dem Display Processor.

Die Ein- und Ausgabeprimitiven werden verwendet, um Operationen auf Text unter der Kontrolle des Programms auszuführen. Der Interpreter führt Programme aus und verwendet dabei die Ein- und Ausgabeoperationen, wenn verlangt. Der Command Dispatcher merkt sich, welches Programm zu jeder möglichen Eingabe gehört. Er liest ein Zeichen von einem Terminal und ruft die zugehörige Funktion auf. Das Library System assoziiert Funktionen mit ihren Namen und Dokumentation und ermöglicht das Laden von Gruppen von Funktionen. Der Display Processor aktualisiert die Ausgabe um die Änderungen an dem Text sichtbar zu machen.

Ursprünglich war Emacs eine Erweiterung für den TECO (Tape Editor and Corrector) Wortprozessor. TECO ist eine Interpretersprache, welche einem die Möglichkeit bietet, auf die internen Datenstrukturen des Interpreters zuzugreifen. Tatsächlich sind dies notwendige Vorraussetzungen, um einen erweiterbaren Editor zu implementieren. Für einen solchen wird eine Interpretersprache benötigt, um auch während der Editor gestartet ist Erweiterungen hinzuzufügen. Die Möglichkeiten in Lisp, Funktionen als Daten zu behandeln, führte dazu, dass die meisten Implementierungen von Emacs einen Lisp-Dialekt als Interpretersprache verwenden.

Die Möglichkeiten zur Erweiterung führte zu einer Vielzahl von Erweiterungen von Emacs. Die einfachste und zuerst implementierte war automatisches Einrücken für Programmiersprachen, insbesondere Lisp. Im Laufe der Zeit wurden sogenannte "Major Modes" für eine Vielzahl von Programmiersprachen entwickelt. Ein Major Mode passt das Verhalten von Emacs an das Editieren einer bestimmten Textart an. Major Modes schliessen einander aus, ein Bearbeitungspuffer kann nur genau einen Major Mode aktiviert haben. Ein Major Mode kann die Bedeutung gewisser Tasten ändern, Syntax-Highlighting für bestimmte syntaktische Konstrukte bereitstellen, automatisch Einrücken und andere Hilfestellungen beim Bearbeiten von Texten bereitstellen wie zum Beispiel kompilieren von Quellcodes direkt aus dem Editor.

Das Emacs Handbuch unterscheidet drei Arten von Major Modes: Eine Menge von Major Modes zum Bearbeiten von normalem Text, wie zum Beispiel der HTML-Mode, Text-mode und LaTeX–Mode. Die zweite Menge von Major Modes sind Modes für spezifische Programmiersprachen, zum Beispiel der Lisp-Mode, C-Mode oder Haskell-Mode. Die dritte Art von Modi sind Modes, die nicht für die Verwendung auf Dateien gedacht sind, sondern für bestimmte Anwendungen in Emacs verwendet werden. Darunter fallen der Dired-Mode, um Verzeichnisse zu verwalten, der IRC-Mode, oder Gnus, ein Mail- und Newsreader für Emacs. Auch eine Vielzahl von Spielen für Emacs fällt in diese Kategorie, darunter zum Beispiel Tetris oder Gomoku.

Lisp-Interpreter existieren für eine Vielzahl von Plattformen. Insbesondere der Display-Processor ist damit auf jeder Plattform mit einem solchen Interpreter implementierbar. Deshalb existieren für eine Vielzahl grafischer und auch text-basierter Oberflächen Implementierungen von Emacs.

1.4 Lisp und Emacs-Lisp

Lisp (LISt Processing language) wurde ursprünglich von Professor McCarthy am Massachusetts Institute of Technology für die Erforschung künstlicher Intelligenz entwickelt. Duzende Lisp-Implementationen wurde seither entwickelt, fast alle mit eigenen Merkmalen und Vorzügen. Viele wurden von Maclisp abgeleitet, welches in den 60er Jahren am MIT entwickelt wurde. Später, 1984 haben die Abkömmlinge von Maclisp einen gemeinsamen Sprachstandard entwickelt, Common Lisp. Inzwischen entwickelten Gerry Sussman und Guy Steele am MIT einen vereinfachten, aber sehr mächtigen Lisp-Dialekt, Scheme. GNU Emacs Lisp ist weitestgehend von Maclisp und

ein wenig von Common Lisp abgeleitet. Ein erfahrener Common Lisp Programmierer wird viele Ähnlichkeiten feststellen. Jedoch mussten eine Vielzahl von Eigenschaften von Common Lisp weggelassen oder vereinfacht werden, um den Speicherbedarf möglichst gering zu halten.[3]

Lisp ist eine Interpretersprache. Obwohl einige Lisp-Compiler existieren, wird Lisp hauptsächlich interpretativ abgearbeitet. Da damit zu einem Lisp-Programm auch gleich der Quelltext vorliegt und Änderungen sofort umgesetzt werden, vereinfacht sich der Entwurf und führt zu einer einfachen Erweiterbarkeit von Lisp-Programmen. Lisp unterscheidet nicht zwischen Daten und Programmtext. Unter anderem diese Uniformität von Quellcode und Daten ermöglicht die Verarbeitung hochstrukturierter symbolischer Daten. Das ursprünglich von McCarthy entwickelte Lisp war annähernd eine Eins-zu-Eins-Umsetzung des Lambda- Kalküls. Auch jetzt spielt die Auswertung von Lambda-Ausdrücken eine bedeutende Rolle in der Programmierung von Lisp-Programmen. Aus der Verwendung des Lambda-Kalküls, welches die Grundlage der Funktionalen Programmierung ist, lässt sich auch das funktionale als das bevorzugte Programmierparadigma bei der Lisp-Programmierung ableiten.

Die meisten Lisp-Dialekte, und so auch Emacs-Lisp, verwenden oder ermöglichen die Verwendung von Seiteneffekten. Ein wichtiges Merkmal von Lisp im Vergleich zu vielen anderen Programmiersprachen, ist, dass die Variablennamen zur Laufzeit erhalten bleiben. Lisp behält die Verbindung zwischen den Namen einer Variable und dem Wert der Variable bei. Für einen erweiterbaren Editor wie Emacs ist dies verbunden mit der Verwendung von globalen Variablen sehr vorteilhaft. Emacs besitzt zum Beispiel eine Variable, die `Comment Start` heisst, und einen String enthalten soll, welcher in einem Text einen Kommentar einleitet. Emacs verwendet die Variable für Operationen auf Kommentaren, wie das Einrücken oder Löschen von Kommentaren. Durch das Ändern dieser Variable kann ein Benutzer das Verhalten dieser Emacs-Funktionen anpassen.

Daten und Programme in Lisp sind aus symbolischen Ausdrücken aufgebaut, die entweder Atome oder Listen sind. Es existieren numerische Atome (Zahlen), Strings (Zeichenketten) und symbolische Atome (Symbole). Eine Liste besteht aus einer öffnenden Klammer gefolgt von beliebig vielen Listenelementen und einer schliessenden Klammer. Ein Listenelement ist eine Liste oder ein Atom.

Die wichtigste Datenstruktur in Lisp ist die Liste. Eine Liste ist entweder die leere Liste (`nil`) oder besteht aus einem Kopf (`car`) und einer Restliste

(`cdr`).

In Lisp gibt es eine Vielzahl von Datentypen. Allerdings sind nicht die Variablen, sondern die Objekte selbst getypt, eine Variable kann jede Art von Lisp-Objekt speichern. Die bereitgestellten Typen umfassen Integers, Funktionen, Zeichen, Zeichenketten, Bitvektoren, Vektoren, Felder, Zahlen, Folgen, Listen und mehr.

Funktionen werden mit `defun` definiert. Eine Funktion kann eine beliebige Anzahl Parameter akzeptieren. Der Funktionskörper kann eine beliebige Folge symbolischer Ausdrücke sein.[5]

Eine wichtige Eigenschaft und Besonderheit von Emacs-Lisp ist die Verwendung von Dynamic Variable Scope. Das heisst, dass die Bindung einer Variablen in allen Aufrufen von Unterroutinen sichtbar ist, es sei denn, die Variable wird neu gebunden. Wird zum Beispiel verwendet:

```
(defun foo1 (x)
  (foo2))
(defun foo2 ()
  (+ x 5))
```

Dann liefert ein Aufruf von `(foo1 2)` 7 zurück, da `foo2` beim Aufruf den Wert von `foo1` für x verwendet.

1.5 Netzwerkprotokoll der Implementierung des Spielservers

Der Spielserver wurde von Dr. Waldmann bereits für einen Wettbewerb 2001 implementiert und generisch genug gehalten, um über diesen Server weitere Zwei-Personen-Spiele zu spielen.

Das Protokoll startet damit, dass der Client nach seinem Namen gefragt wird:

```
S: Wer_bist_du
C: Ich_bin "test"
```

Hierbei sind die von dem Server geschickten Zeilen mit einem “S:” gekennzeichnet, die Antwort des Client hingegen mit einem “C:”. Diese Bezeichnung wird im Laufe diese Arbeit beibehalten.

Im Anschluss gibt der Server dem Client die Parameter des Spielfeldes bekannt:

```
S: Das_Spielfeld_ist (Sportplatz{breit=15,hoch=19,ball=H10})
C: OK
S: Die_Spielzeit_ist (Zeit{main=20,byoyomi=60,moves=3})
C: OK
S: Dein_Gegner_ist "Test-Gegner"
C: OK
S: Du_beginnst
C: OK
```

Hier werden dem Client die Art des Spiels (Sportplatz) sowie die Ausmasse des Spielfeldes sowie die Spielzeit mitgeteilt, und der Name des Gegners. Der Client bestätigt jeweils mit "OK". Schliesslich teilt der Server dem Client mit, ob er oder sein Gegner den ersten Zug machen darf.

Nachdem die allgemeinen Daten ausgetauscht wurden, fordert der Server die Spieler nun jeweils abwechselnd auf, einen Spielzug zu schicken.

```
S: Die_Restzeit_ist(Zeit{main=20,byoyomi=60,moves=3})
C: OK
S: Wo_ziehst_du
```

Jetzt hat der Client mit einem gültigen Zug zu antworten. Es exitieren zwei Arten von Zügen, Spieler setzen und Springen.

Es folgt ein Beispiel für das Setzen eines Spielers

```
C: Ich_ziehe (Place H9)
```

und für das Überspringen von Spielern:

```
C: Ich_ziehe (Jump [H10,H12])
```

Nachdem ein Spieler einen Zug abgeschickt hat, übermittelt der Server den Zug an den Gegenspieler, wonach dieser die Möglichkeit hat, einen Zug abzugeben.

```
S: Der_andere_zieht(Place H11)
C: OK
```

[7]

2 Phutball Major Mode

In diesem Abschnitt wird auf den eigentlichen Major Mode eingegangen. Im ersten Teil wird die Bedienung beschrieben. Im zweiten Teil wird das Vorgehen bei der Implementierung beschrieben.

2.1 Bedienung

Nach dem Start des Major Modes mit `M-x phutball-mode` wird man aufgefordert, die Adresse des Spielservers, anschliessend die des TCP-Ports, und darauf seinen Spielernamen einzugeben. Hat man das getan, sieht man das Spielfeld. Ein freies Feld ist als Punkt, ein Spieler als X und der Ball als @ dargestellt. Abbildung 1 zeigt eine normale Spielsituation im Major Mode. Die Bewegung des Cursors geschieht standardmässig mit den Pfeiltasten. Ein Spieler wird gesetzt mit `RET`. Ein Sprungvorgang wird mit `J` eingeleitet, anschliessend wird durch Eingabe einer Ziffer ein Teilsprung in die gewählte Richtung durchgeführt. Abbildung 2 zeigt eine Darstellung einer Sprungsituation.

Ausserdem erkennt man an den Darstellungen, dass die Randlinien mit dargestellt werden, die Seitenlinie durch ein @ und die Torlinien durch Linie 0 bzw. *Linienzahl* + 1.

2.2 Die Implementierung

In diesem Teil des Abschnitts wird die Implementierung beschrieben. Zu Beginn wird die Grundstruktur des Major Modes erläutert, anschliessend einzelne ausgewählte Funktionen und Designentscheidungen diskutiert.

2.2.1 Grundstruktur

Folgende Schritte sind zu befolgen bei der Erstellung eines Major Modes [1][2]:

1. Wählen eines *Namen* für den Major Mode. Der Name dieses Modes ist `phutball`.

2. Erstellen einer Datei mit der Endung `.el`, welche den Code für den Major Mode enthält.

```
  @  A  B  C  D  E  F  G  H  I  J  K  L  M  N  O  @

20.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
19.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .        Zug: 22
18.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .        Zeit: 12
17.  .  .  .  .  .  .  .  .  .  X  .  .  .  .  .  .
16.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
15.  .  .  .  .  .  .  .  .  .  X  .  .  .  .  .  .
14.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
13.  .  .  .  .  .  .  .  .  X  .  .  .  .  .  .  .
12.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
11.  .  .  .  .  .  .  X  .  .  .  .  .  .  .  .  .
10.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
9 .  .  .  .  .  .  X  .  .  .  .  .  .  .  .  .  .
8 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
7 .  .  .  .  .  .  X  .  .  .  .  .  .  .  .  .  .
6 .  .  .  .  .  .  X  .  .  .  .  .  .  .  .  .  .
5 .  .  .  .  .  .  .  X  .  .  .  .  .  .  .  .  .
4 .  .  .  .  .  .  .  .  @  .  .  .  .  .  .  .  .
3 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
2 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
1 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
0 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .

Spieler: Emacs-Test vs. $Id: Flankengott.hs,v 1.5 2002/04/02 \
 22:04:09 joe Exp $\,breit: 5,tief: 4(from localhost)
```

Abbildung 1: Darstellung einer Spielsituation

```
  @  A  B  C  D  E  F  G  H  I  J  K  L  M  N  O  @

20.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
19.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .       Zug: 77
18.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .       Zeit: 66
17.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
16.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
15.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
14.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
13.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
12.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
11.  .  .  .  .  .  .  .  .  X  .  .  .  .  .  .  .  .
10.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
9 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
8 .  .  .  .  .  .  .  .  .  8  .  .  .  2  .  .  .  .
7 .  .  .  .  .  .  .  .  .  .  X  .  X  .  .  .  .  .
6 .  .  .  .  .  .  .  .  .  .  .  0  .  .  .  .  .  .
5 .  .  .  .  .  .  .  .  .  .  X  .  .  .  .  .  .  .
4 .  .  .  .  .  .  .  .  .  X  .  .  .  .  .  .  .  .
3 .  .  .  .  .  .  .  .  6  .  .  .  .  .  .  .  .  .
2 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
1 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
0 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .

Spieler: Emacs-Test vs. $Id: Flankengott.hs,v 1.5 2002/04/02 \
 22:04:09 joe Exp $\,breit: 5,tief: 4(from localhost)
```

Abbildung 2: Sprungsituation im Major Mode

3. Definieren einer Variablen `phutball-mode-hook`. Diese wird die Funktionen enthalten, die beim starten des Modes ausgeführt werden.

4. Definieren einer mode-spezifischen Keymap[1]. Die Variable `phutball-mode-map` wird mit dieser Keymap belegt. Eine Keymap wird wie folgt erzeugt:

```
(defvar phutball-mode-map nil
   "Keymap fuer phutball mode")

(if phutball-mode-map nil
  (setq phutball-mode-map (make-keymap))
(define-key phutball-mode-map keysequence command)
...)
```

5. Falls erforderlich, wird eine mode-spezifische Syntaxtabelle erzeugt und eine Variable `phutball-mode-syntax-table` damit belegt. Für unseren Mode benötigen wir keine Syntaxtabelle.

6. Definieren eines Kommandos `phutball-mode`. Das ist das Major-Mode-Kommando, und benötigt keine Argumente. Wenn es ausgeführt wird, sollte es den aktuellen Buffer[2] in einen phutball-mode Buffer überführen[3], indem es folgende Schritte durchführt:

   (a) Es muss `kill-all-local-variables` aufrufen, womit die Definitionen aller lokalen Buffervariablen entfernt werden. Dies schaltet alle Major- und Minor-Modes, welche vorher aktiv waren, aus.

```
(kill-all-local-variables)
```

   (b) Es muss die Variable `major-mode` auf `phutball-mode` setzen.

```
(setq major-mode 'phutball-mode)
```

[1] Eine Keymap ordnet Tastenkombinationen Funktionen zu.

[2] Ein Buffer ist ein Objekt in Emacs, welches den Text enthält. Jeder Buffer hat einen Namen, und jedes Fenster stellt einen Buffer dar. Es existiert stets nur ein aktiver Buffer. Auf den Inhalt dieses Buffers operieren die ausgeführten Kommandos.

[3] In jedem Buffer ist genau ein Major Mode aktiv.

(c) Es muss die Variable `mode-name` auf eine kurze Zeichenreihe setzen, welche den Mode beschreibt.

```
(setq mode-name "Phutball")
```

(d) Es muss die modespezifische Keymap installieren, falls diese existiert, indem `use-local-map` aufgerufen wird.

```
(use-local-map phutball-mode-map)
```

(e) Es muss die Hook[4]-Funktionen ausführen mit `run-hooks`.

```
(run-hooks 'phutball-mode-hook)
```

7. Funktionen des Modes müssen "bereitgestellt" werden durch Aufrufen von `provide`.

```
(provide 'phutball)
```

Die Major-Mode Funktion ist in Abbildung 3 dargestellt.

2.2.2 Netzwerkfunktionen

Lange Zeit war es nicht möglich, unter Emacs direkt Netzwerkfunktionalität zu verwenden. Emacs Lisp kann seit Emacs 19 TCP-Verbindungen verwenden. Zum Verständnis von den Netzwerkfunktionen ist es nötig, von Emacs Lisp erzeugte und verwaltete Prozesse zu verstehen. Ein Prozess ist ein Raum, in welchem ein Programm ausgeführt wird. Emacs läuft in einem Prozess. Elisp-Programme können Kindprozesse erzeugen. Es existieren synchrone und asynchrone Prozess. Wird ein synchroner Prozess erzeugt, wartet das Lisp-Programm auf das Beenden dieses Prozesses, bevor es mit der Ausführung fortfährt. Asynchrone Prozesse können parallel zu dem erzeugenden Lisp-Programm laufen.

[4]Ein Hook stellt die Möglichkeit zur Verfügung, benutzerdefinierte Funktionen an definierten Stellen, zum Beispiel dem Öffnen einer Datei oder dem Starten eines Major Modes, auszuführen.

```
(defun phutball-mode ()
  "Major mode for playing phutball games over a network server."
  (interactive)
  (switch-to-buffer (get-buffer-create "*Phutball*") t)
  (erase-buffer)
  (kill-all-local-variables)
  (setq major-mode 'phutball-mode)
  (setq mode-name "Phutball")
  (run-hooks 'phutball-mode-hook)
  (phutball-goto-xy 1 1)
  (use-local-map phutball-mode-map)
  (setq buffer-read-only t)
  (setq phutball-server (read-from-minibuffer "Server: "
                                      phutball-default-server))
  (setq phutball-port (string-to-number (read-from-minibuffer "Port: "
                                  (number-to-string phutball-default-port))))
  (setq phutball-player-name (read-from-minibuffer "Your Name: "
                                      phutball-player-name))
  (message "Stand by while connecting")
  (turn-on-font-lock)
  (phutball-open-connection)
  (run-at-time 0 1 'phutball-display-time)
  (make-local-variable 'font-lock-defaults)
  (setq font-lock-default '(phutball-font-lock-keywords))
)
```

Abbildung 3: Die Major-Mode-Funktion.

```
(defun phutball-open-connection ()
  (open-network-stream "phutball-socket" "phutball-socket-buffer"
                       phutball-server phutball-port)
  (set-process-filter (get-process "phutball-socket") 'phutball-filter))
```

Abbildung 4: Herstellen einer Verbindung zum Phutball Server. `phutball-server` und `phutball-port` sind globale Variablen, die beim Start des Major Modes gesetzt werden.

Eingabe an Prozesse geschieht durch spezielle Elisp-Funktionen, zum Beispiel `process-send-string`. Es gibt zwei Wege, um die Ausgabe, die ein Prozess auf seine Standardausgabe schreibt, zu empfangen. Die Ausgabe kann in einen Buffer eingefügt werden, oder eine Funktion kann aufgerufen werden, welche mit der Ausgabe Operationen ausführt.

Die Filterfunktion eines Prozesses ist eine Funktion, welche die Standardausgabe des zugehörigen Prozesses empfängt. Wenn ein Prozess einen Filter besitzt, wird jede Ausgabe des Prozess an die Filterfunktion übergeben. Eine Filterfunktion akzeptiert zwei Argumente: den assoziierten Prozess und den empfangenen String. Die Filterfunktion hat dann die Möglichkeit, beliebiges mit den empfangenen Daten zu tun.

Eine Netzwerkverbindung in Emacs Lisp verhält sich wie ein Kindprozess und wird auch durch ein Prozessobjekt repräsentiert. Da die Netzwerkverbindung aber kein echter Kindprozess ist, kann man ihm keine Signale schicken oder ihn killen[5], sondern nur Daten von ihm empfangen oder senden.[3]

Der Phutball-Major-Mode realisiert die Netzwerkkommunikation durch eine TCP-Verbindung mit einer Filterfunktion, welche das Protokoll des Phutball-Spielservers wie in Kapitel 1.5 implementiert. Das Erzeugen der Verbindung ist in Abbildung 4 dargestellt.

Die Filterfunktion fügt den empfangenen Text erst in einen temporären Buffer ein, und führt noch einige syntaktische Operationen auf den empfangenen Daten durch, wie das entfernen von Leerzeichen und von Kommentaren. Der Grund, warum die Daten nicht sofort verarbeitet werden können liegen darin, dass die Daten nicht zeilenweise ankommen, sondern, da sie über ein Netzwerk übertragen werden, in beliebig kleinen Teilen. In dem Buffer werden aus diesen Teilen ganze Kommandos erzeugt, die dann verarbeitet

[5]Das "Killen" von Prozessen bedeutet die sofortige Beendigung desselben, meist realisiert durch das Senden von einem Signal auf Betriebssystemebene.

```
(defun phutball-filter (proc string)
  ...
  ((re-search-forward "Dein_Gegner_ist[^\n]*" nil t)
    (progn
      (copy-to-register 'a (+ 17 (match-beginning 0))
                        (- (match-end 0) 1))
      (with-current-buffer (get-buffer "*Phutball*")
        (setq phutball-opponent-name (get-register 'a)))
          (process-send-string (get-process "phutball-socket")
                               "OK\n")
        (goto-char (line-end-position))
        (insert "\nOK\n")))
  ...)
```

Abbildung 5: Auszug aus der Filterfunktion

```
(defun phutball-goto-xy (x y)
  ``Move point to square at X, Y coords.''
  (let ((inhibit-point-motion-hooks t))
    (goto-line (+ phutball-y-offset
                  (* (- phutball-field-height(- y 2))
                     phutball-square-height))))
  (move-to-column (- (* (+ 1 x) phutball-square-width)
                     1)))
```

Abbildung 6: Funktion zur Bewegung des Points in Phutball-Koordinaten

werden können. Einen Auszug stellt die Abbildung 5 dar. In dem Buffer wird nach *Dein_Gegner_ist* gesucht, und dann der Name des Gegners kopiert und die Variable *phutball-opponent-name* gesetzt. Anschliessend wird dem Spielserver mit *OK* das Kommando bestätigt.

2.2.3 Funktionen zur Bewegung auf dem Spielfeld

Das Spielfeld kann beliebige Grössen annehmen. Deshalb sind Funktionen zur Bewegung auf dem Spielfeld und zur Ermittlung der Koordinaten nötig. Es existieren Funktionen, zur Bewegung des Cursors in jede der vier Richtungen auf dem Spielfeld. Ausserdem existiert eine Funktion zur Bewegung

des Points[6] an beliebige Koordinaten. Diese Funktion ist in Abbildung 6 dargestellt.

2.2.4 Operationen auf dem Spielfeld

Wie in Kapitel 1.2 dargestellt, existieren zwei Operationen auf dem Spielfeld, Spieler setzen und Springen. Das Setzen eines Spielers geschieht einfach durch Ersetzen des freien Feldes, dargestellt durch einen Punkt, durch ein von einem Spieler besetztes Feld, dargestellt durch ein "X". Wird der Spieler von dem Gegner gesetzt, werden diese Operationen durch die Filter-Funktion durchgeführt. Ansonsten wird der Spieler durch die interaktive Funktion `phutball-set-player-i` gesetzt. Diese sendet auch die Koordinaten des gesetzten Spielers an den Spielserver.

Es existieren ebenso zwei Sprungfunktionen, eine interaktive, die von dem mit dem Major Mode spielenden Spieler aufgerufen wird, und eine Funktion, welche Sprünge, die durch den Gegner durchgeführt wurden, ausführt. Die Funktion `phutball-jump` akzeptiert einen String, wie er von dem Phutball-Server übertragen wird, und führt die notwendigen Operationen auf dem Spielfeld aus. Die Funktion `phutball-jump-i` führt einen interaktiven Sprung des Benutzers aus. Dazu werden jeweils alle direkten Sprünge ermittelt und an die mögliche Endposition eine Ziffer gesetzt. Durch Eingabe der Ziffer wird der Teilsprung durchgeführt, und die Funktion rekursiv aufgerufen, bis der Benutzer keinen weiteren Teilsprung wünscht.

2.2.5 Hilfsfunktionen

Eine Vielzahl von Hilfsfunktionen sind nötig. Die beudeutendsten sind die Konvertierungsfunktionen. Ein Emacs-Buffer verwendet ein Koordinatensystem welches von dem Phutball-Koordinatensystem verschieden ist. Ausserdem werden in dem Major Mode zur Darstellung der Phutball-Koordinaten ein Tupel von zwei Zahlen verwendet, der Spielserver verwendet jedoch ein Tupel aus einem Buchstaben und einer Zahl. Es sind also zwei Transformationen nötig. Die Transformation von der Darstellung als Tupel zweier Zahlen in Emacs-Koordinaten wurde bereits diskutiert, dies erledigt die Funktion `phutball-goto-xy` implizit. Die inverse Transformation wird von den beiden Funktionen `phutball-point-x` und `phutball-point-y` erledigt.

[6]Der Point gibt die aktuelle Position innerhalb eines Buffers an. Lokale Kommandos operieren an der Stelle des Points.

```
(defun phutball-convert-coords (l)
  "Converts phutball-string-coordinates to phutball-number-coordinates."
  (let ((temp (string-to-list l)))
    (setq a (- (car temp) 64))
    (if (> (length temp) 2)
        (setq b (+ (* 10 (- (car (cdr temp)) (string-to-char "0")))
                   (- (car (cdr (cdr temp))) (string-to-char "0"))))
      (setq b (- (car (cdr temp)) (string-to-char "0")))))
  (list a b))
(defun phutball-convert-coords-backward (x y)
  "Converts phutball-number-coordinates to phutball-string-coordinates"
  (let ((as (string (- (+ x ?@) 0))))
    (concat as (number-to-string y))))
```

Abbildung 7: Konvertierungsfunktionen

Die Funktionen zur Transformation der Phutballkoordinaten von der Repräsentation des Spielservers in die Repräsentation in diesem Major Mode und umgekehrt werden in der Abbildung 7 gezeigt.

Die Funktion `phutball-display-field` wird aufgerufen, sobald die Filterfunktion die Parameter des Spielfeldes von dem Spielserver erfahren hat, und sie stellt das Koordinatensystem dar.

`phutball-new-game` ermöglicht es dem Benutzer, ein neues Spiel zu beginnen. Das alte Spiel wird dabei zerstört.

Zuletzt existieren noch drei weitere Informationsfunktionen, `phutball-set-playername`, `phutball-set-movecount` und `phutball-display-time`. Die erste zeigt die Namen der beiden Spieler in einer Statuszeile unter dem Spielfeld, und wird von der Filterfunktion aufgerufen, nachdem diese die Namen der Spieler ermittelte. Die zweite zeigt die Anzahl der bisher durchgeführten Spielzüge des Spielers. Die Funktion `phutball-display-time` wird pro Sekunde einmal aufgerufen[7] und zeigt die verbleibende Restzeit des Spielers an, und tut nichts, wenn der Gegner am Zug ist.

[7]Die Funktion, um eine andere Funktion in regelmässigen Intervallen aufzurufen, ist `run-at-time`, siehe dazu auch Abbildung 3.

3 Schluss

Es wurden die Verwendungsmöglichkeiten eines Major Modes für das Zweipersonenspiel Phutball dargelegt. Mit dem Major Mode existiert ein weitestgehend plattformunabhängiges Frontend zum Phutball Spielen. Durch die Quelloffenheit und die Verwendung von Lisp ergeben sich vielfältige Möglichkeiten zur Nachnutzung und Weiterentwicklung. Die nützlichste Erweiterung wäre die Integration eines Computerspielers in den Major Mode, so dass die Verbindung zu einem Phutball-Server überflüssig wäre. Es bietet sich an, eines der am besten platzierten Programme des Programmierwettbewerbs beziehungsweise deren Algorithmen zu verwenden. Da die Quellen der Spielprogramme offen liegen und zum Teil auch funktionale Sprachen für die Implementierung verwendet wurden, stellt die Portierung auf Lisp keine grossen Schwierigkeiten dar. Existiert erst ein Major Mode, welcher ohne einen Spielserver verwendbar ist, besteht die Möglichkeit, den Major Mode in die offiziell von der Free Software Foundation herausgegebene Version von Emacs zu integrieren. Dies würde zu einem höheren Bekanntheitsgrad von Phutball, und schliesslich auch zu stärkeren Spielprogrammen führen.

Kleinere Erweiterungen umfassen die Integration einer Maussteuerung, die Einbindung von Grafiken und Audioausgaben.

Der Quellcode des Major Modes für Phutball sowie diese Dokumentation ist auf der Webseite des Autors unter http://leechuck.de/phutball zu finden.

Literatur

[1] Bob Glickstein, Writing GNU Emacs Extensions, O'Reilly, 1997

[2] Richard Stallman, GNU Emacs Manual, Free Software Foundation, 2002, http://www.gnu.org/manual/emacs/index.html

[3] Ben Lewis, Dan LaLiberte, Richard Stallman and the GNU Manual Group, GNU Emacs Lisp Reference Manual, Free Software Foundation, 1998, http://www.gnu.org/manual/elisp-manual-20-2.5/

[4] Johannes Waldmann, GAOTenBlatt, Seite 21ff, GAOS e.V., 2002

[5] Gerd Herzog, Peter Wazinski, KI-Programmierung in LISP - Kursunterlagen, Universität des Saarlandes, 1994, http://www.dfki.uni-sb.de/ flint/dokuments/kifs94/kifs94.html

[6] Richard Stallman, EMACS: The Extensible, Customizable Display Editor, Free Software Foundation, http://www.gnu.org/software/emacs/emacs-paper.html

[7] Johannes Waldmann, Programmierwettbewerb Phutball, Sommersemester 2002, Universität Leipzig, 2002, http://www.informatik.uni-leipzig.de/ joe/wettbewerb/phutball