Witchcraft - Technische Dokumentation

Andreas Textor und Ralph Erdt

(Programmieren 3 Semesteraufgabe WS06/07)

16. Januar 2007

Autoren: Ralph Erdt (Matr.Nr. 351266) und Andreas Textor (Matr.Nr. 751375)

Wir erklären hiermit, dass das vorliegende Programm ausschließlich von Ralph Erdt und Andreas Textor und ausschließlich unter der Nutzung der erlaubten Hilfsmittel erstellt wurde.

1 Hinweise zur technischen Dokumentation

Dieses Dokument ist die Technische Dokumentation; für die Spielanleitung starten Sie bitte das Spiel und wählen im Menü „Hilfe - Anleitung“.
Die technische Dokumentation wird ergänzt durch die generierte Javadoc-Dokumentation, die die Klassen und deren Methoden im Einzelnen beschreibt, und die im Verzeichnis */dokumentation/javadoc/* zu finden ist.
Nomenklatur: Alle Klassennamen beginnen mit „T“, alle Interfacenamen mit „I“. Ausnahmen sind SpielApplet [4], Main [3] und Game [1] (letzteres ist das Interface, das in der Aufgabenstellung verlangt wird).

2 Spielidee

2.1 Grundideen

Witchcraft ist ein „Sidescroller“, der Spieler fliegt nach rechts, während die Gegner nach links fliegen. Der Spieler steuert eine Hexe auf einem fliegenden Besen, die ankommende Gegner mit Zaubern und schweren Waffen abschießen kann. Die Gestalt der Hexe sowie die Kulisse sind von den Scheibenwelt-Romanen von Terry Pratchett inspririert. Für die Gegner soll gelten „Masse statt Klasse“, d.h. die Schwierigkeit ergibt sich durch die hohe Anzahl von Gegnern, von denen jeder einzelne alleine leicht zu besiegen ist.
Der Spieler verwendet zum Steuern der Hexe die Pfeiltasten, und zur Kontrolle des Fadenkreuzes die Maus.

2.2 Waffen

Die Waffen benötigen keine Munition, aber der Spieler hat einen „Magie“-Wert, der durch jedes Abfeuern einer Waffe um einen bestimmten Wert reduziert wird. Der Magie-Wert lädt sich konstant von selbst auf, aber nicht so schnell, dass man mit jeder Waffe ununterbrochen feuern könnte. Der Magie-Wert wird im Spiel durch einen blauen Balken am unteren Bildrand angezeigt.

3 Quellen

Alle Quellcodes des Spiels wurden von Ralph Erdt und Andreas Textor geschrieben. Alle Bilder (Titel- und Hintergrundbild, Gegner-, Waffen- und Item-Grafiken sowie die Grafiken der Levelsegmente) wurden von Ralph Erdt und Andreas Textor gemalt mittels Inkscape und GIMP (GIMP-Quelldateien, die sich von den im Spiel verwendeten Grafiken unterscheiden (diese befinden sich im Verzeichnis bilder/), befinden sich im Verzeichnis gfxsrc/). Hilfestellungen in der Bedienung des GIMP sowie Überarbeitung des Titelbildes, sowie die Musik des Spiels stammen von Martin Textor. Die verwendeten Soundeffekte stammen aus verschiedenen freien Quellen im Internet.

4 Die wichtigsten Klassen

- *TVektor* [32]. Da dies ein Spiel im 2-dimensionalen Raum ist, sind alle Koordinaten und Bewegungen durch Vektoren dargestellt. Die Klasse TVektor enthält neben den x- und y-Koordinaten auch alle benötigten Rechenoperationen bis hin zur Winkelberechnung zwischen zwei Vektoren. TVektor ist die mit Abstand am häufigsten verwendete Klasse, die Exemplarvariablen x und y sind daher zur Vereinfachung als public deklariert, und benötigen keine get- und set-Methoden.

- *TObjekt* [16]. Fast alle Spielobjekte, die auf dem Spielfeld auftauchen, sind vom Typ TObjekt oder einer Unterklasse. Die Klasse stellt Vektoren zur Verfügung für Koordinaten

(fkoord), Größe (fdim) und gerichteter Bewegung (fgeschw) eines Objektes. Ebenfalls vorhanden sind Methoden zur Bewegung und Kollisionsabfrage, es sind aber keine Methoden zur Darstellung vorhanden.

- *TAnimation* [5] ist ein Wrapper um eine Liste von BufferedImages sowie einem Zahlenwert, der angibt, wie lange jeder Animationsframe angezeigt werden soll, bevor der nächste ausgewählt wird. Hat ein Objekt nur eine unbewegliche Darstellung, so enthält die Bilderliste nur einen Eintrag.
  Welcher Animationsframe der aktuell anzuzeigende ist, entscheidet dabei alleine TAnimation; Objekte, die TAnimation verwenden, müssen sich darum nicht kümmern. TAnimation wird außerdem dazu verwendet, vorab gedrehte Bilder von Geschossen zu speichern, dabei ist jedes gedrehte Bild ein Eintrag in der Bilderliste und wird über die Angabe der Ausrichtung (Drehwinkel) des entsprechenden Geschosses ausgewählt (siehe auch Technische Probleme).

- *TBildObjekt* [7]. Jedes durch ein Bild oder eine Animation repräsentierte Spielobjekt ist vom Typ TBildObjekt, welches von TObjekt erbt und ein Objekt vom Typ TAnimation enthält. Alle Lebewesen (d.h. Spieler und Feinde), Items und Geschosse sind TBildObjekte, außerdem auch Partikel, falls die Einstellung „Hohes Grafikdetail“ gewählt wurde (siehe auch Technische Probleme).

- *TAnzeige* [6] erbt von JPanel und ist die spielverwaltende Klasse und enthält Listen zur Verwaltung der Spielobjekte (Gegner, Geschosse, usw.). Hier werden Aufrufe zum Laden von Ressourcen (Konfiguration, Bilder, Sounds, Level) getätigt, Spielobjekte werden bewegt und gezeichnet. Darüber hinaus enthält TAnzeige alle Timer, Maus- und Tastaturlistener.

5 Sonstige Klassen

- *TPartikelVerwaltung* [23]. Alle Effekte wie Funken, Explosionen und Rauch bestehen aus vielen kleinen Partikeln und werden von dieser Klasse verwaltet. Nach aussen hin sichtbar sind nur die Methoden startEffekt(), der man den gewünschen Effekttyp (Enum Partikel) und einen Koordinaten-Vektor, der angibt, wo der Effekt stattfinden soll, übergibt, und eine Methode zum Zeichnen.

- *TConfig* [8] lädt alle Ressourcen aus der Konfigurationsdatei *config/witchcraft.xml*. Dort sind Bilder und Animationen, Waffen und Geschosse sowie Sounds definiert. Nur Level- und Gegnerdefinitionen werden nicht aus dieser Konfigurationsdatei geladen (siehe Beschreibung von TLevel).

- *TLevel* [14]. Ein Level besteht aus einer Liste von „Scheiben“, sog. Levelsegmenten, die hier als innere Klasse realisiert sind. Diese werden einfach „weitergeschoben“, je weiter der Spieler im Level fortschreitet. Dabei hat jedes Levelsegment eine Liste von Feinden, die mit dem Laden des Levels bereits erzeugt werden, und die aktiviert (d.h. in die Feindesliste in TAnzeige eingekettet zur Darstellung und Bewegung) werden, sobald das Levelsegment in den sichtbaren Bereich auf dem Bildschirm reinscrollt.
  Außerdem kann ein Levelsegment eine „Meldung“ enthalten, ein Bild, das beim Aktivieren des Levelsegments zentriert dargestellt wird und das z.B. Hinweise zur Bedienung des Spiels oder zur Story enthalten kann.

- *Main* [3] ist die Klasse, die als erstes gestartet wird: Main erzeugt ein Fenster, eine Menü mit MenuItems sowie eine TAnzeige, und fügt alles zusammen. Die MenuItems sind mit ActionListenern verbunden, die entsprechende Methoden in TAnzeige aufrufen (z.b. „Spiel - Neu“).

- *Lebewesen und Feinde.* Lebewesen sind alle Spielobjekte mit einem „Leben“, außerdem sind hier die Trefferberechnungen eingebaut. Jedes Lebewesen hat eine „Seite“, für die es kämpft (0 = Spieler, 1 = Feind, evtl. noch erweiterbar) und Geschosse, die von einer Seite abgefeuert werden, sollen nur Lebewesen der anderen Seite treffen können. TSpieler [29] erbt von TLebewesen [13] und fügt alle Spieler-spezifischen Variablen und Methoden hinzu (z.B. den Magie-Wert oder gesonderte Bewegung). Ebenfalls von TLebewesen erbt TFeind [9]; ein Feind ist ist eine sich autonom bewegende und schiessende Einheit. Im Gegensatz zum Spieler hat jeder Feind nur eine Waffe.
  Jeder Feind hat eine Referenz auf ein TFeindDef [10]-Objekt, das alle Eigenschaften der Feinde dieses Typs zusammenfasst. Hier kann auch ein „Muster“ definiert sein, eines von mehr als 20 festprogrammierten Bewegungsmustern, die ein Fein annehmen kann (Verschiedene lineare, zickzak, sinus- und kreisförmige Bewegungen).

- *TSavegame* [26] Um die Savegamedateien simpel zu halten, und um den Spieler nach dem Laden nicht direkt in eine unüberschaubare Situation zu werfen, werden dort ausser den essentiellen Spielerinformationen (Leben, Score usw.) nur der Fortschritt des Spielers im Level gespeichert. Beim Laden eines Savegames wird dann vor dem nächsten zu aktivierenden Levelsegment eine Bildschirmbreite „Leersegmente“ gehängt. Dadurch erscheinen Gegner in der richtigen Reihenfolge und Geschwindigkeit, und der Spieler hat eine faire Chance nach dem Laden nicht gleich abgeschossen zu werden.

- *Waffen, Geschosse und Items.* Eine Waffe ist eine Sammlung verschiedener Eigenschaften, z.B. Geschwindigkeit und Schadenswirkung der zugehörigen Geschosse.
  Ein Geschoss ist eine Instanz der Waffe, z.B. die Rakete (TRakete [25]). Da die Boundingbox bei länglichen Geschossen größer wird, aber ein Geschoss beim schrägen Flug (z.B. um 45 Grad gedreht) den Gegner nicht leichter treffen soll, als wenn es horizontal oder vertikal fliegt, wurde bei Geschossen eine separate Hitbox zugefügt. Diese ist quadratisch und hat eine Kantenlänge von `min(Geschosslänge, Geschossbreite)` und wird mittig im gedrehten Geschoss platziert. Die Rakete schwenkt „langsam“ auf den Zielkurs ein, und beschleunigt dabei. Außerdem gibt es Homing-Missiles, die das Ziel verfolgen, bis dieses zerstört oder verschwunden (aus dem Bildschirm geflogen) ist. Dann fliegen sie in die aktuelle Richtung weiter. Ein Spezialgeschoss ist das Geschoss der „Strahlenwaffen“ (TRail [24]), die einen „instant Hit“ verursachen. Es gibt also kein fliegendes Geschoss, die grafische Darstellung ist eine Linie zwischen dem Lebewesen, welches gefeuert hat, und dem getroffenen Ziel. Diese Strahlen haben eine „TTL“ (Time to Live), damit sie einen Moment sichtbar bleiben.
  Items [12] sind vom Spieler einsammelbare Gegenstände, die von besiegten Feinden fallen gelassen werden. Items können Waffen oder z.B. die Java-Tasse (Health-Bonus) sein.

- *Sonstige.* Alle bisher nicht erwähnten Klassen sind entweder Hilfsklassen (wie z.B. TPaar [18] - ein generisches Paar zweier Werte) oder für die Spiellogik nicht interessant und daher hier nicht weiter ausgeführt (z.B. TOptionenFenster [17] - der Einstellungsdialog, der beim Start eines neuen Spieles angezeigt wird). TSharedObjects [27] wird im Abschnitt Technische Probleme erläutert.

6 Technische Probleme und Lösungen

- *Viele Objekte und langsame Grafikausgabe.* Da Java (leider?) nicht die Möglichkeit bietet, Speicherverwaltung zumindest stellenweise von Hand zu erledigen, wird nach dem Laden der Ressourcen der Garbage-Collector aufgerufen, damit im Ladeprozess erzeugte und jetzt nicht mehr benötigte Objekte entsorgt werden können, und dieser Vorgang nicht das Spiel stört (wo er nämlich ansonsten zu einem Ruckeln am Anfang des Spiels führen würde). Da die Grafikausgabe speziell von Bildern mit Alphakanal (d.h. im RGBA Format, Bilder mit

echter Transparenz) sehr langsam ist, wurde die Option eingebaut, das Spiel mit „niedrigen Grafikdetails“ zu Spielen (d.h. kein Hintergrundbild, und programmierte Partikel anstatt Partikeln, die BildObjekte zur Darstellung verwenden). Außerdem lässt sich die Musik im Optionenmenü einschalten - Standardmässig ist sie abgeschaltet.
Außerdem wurde, um das Spiel zu beschleunigen, darauf geachtet, möglichst wenige Spielobjekte neu zu erzeugen während der Laufzeit. Teure Operationen wie das Drehen der Geschossgrafiken wird zur Ladezeit erledigt: Ein Geschoss wird 31 Mal gedreht und jedes neue Bild als Frame in die Bilderliste eines TAnimationsobjektes eingekettet.

- *Tastaturfokus im Applet.* Die der Tastaturfokus im Applet stark browserabhängig ist, die Tastatureingabe für das Spielkonzept aber essentiell ist, wird in der Appletversion des Spiels zuächst nur ein Knopf angezeigt. Erst das Klicken auf den Knopf öffnet das eigentliche Spiel in einem neuen Fenster, wie bei der Applikationsversion.

- *Viele Klassen benötigen Zugriff auf bestimmte Funktionalität.* Da einige Klassen wie die für Soundausgabe oder Partikelverwaltung von vielen Klassen Zugriff benötigen, und ein Herumreichen von Referenzen auf Objekte dieser Klassen in Methoden, die selber den Zugriff vielleicht gar nicht benötigen sondern nur weiterreichen den Code unnötig kompliziert und verwirrend machen würde, gibt es die Klasse TSharedObjects [27]. Hier können sich die verbreitet benötigtigen Klassen registrieren, und Anfragende können sich durch eine einfache get-Methode Referenzen auf diese Objekte abholen. Diese Funktionalität ist in Klassenmethoden realisiert, da ein Objekt von TSharedObjects sinnfrei wäre.

- *Sound.* Aus Geschwindigkeitsgründen werden die Sounds beim Starten des Spiels geladen. Wird für jede Datei ein AudioClip verwendet, kann immer nur eine Instanz eines Sounds zur selben Zeit gespielt werden. Deswegen wird ein Pool von Sounds angelegt, von denen immer der nächste freie zum Spielen ausgewählt wird. Die Ausnahme bildet die Musik, da hiervon auf jeden Fall immer nur ein Exemplar zur selben Zeit abgespielt wird. Abgesehen davon würden mehrere Exemplare der Musikdatei eine unnötige Menge an Heap verschwenden.

- Es werden viele *statische Listen* der verschiedensten Objekte benötigt. (Hier ist nicht die Rede von den Listen, die in TAnzeige sind, und die aktuellen Objekte enthalten.)
Z.B. Werden statische Listen für die vorhandenen Animationen oder verfügbaren Waffen benötigt. Da diese statisch sind (beim Starten werden sie aufgebaut, dann nur noch abgefragt) haben wir einige dieser Liste in die dazugehörige Klasse geschrieben. Z.B. ist die Liste der Waffen in TWaffe.

- *Schreiben von XML-Savegames.* Es wurde das Problem beobachtet, dass beim Speichern der Savegames als XML unter Windows das Encoding der Dateien Latin1/ISO-8859-1 war, obwohl die Einstellung für UTF-8 getroffen wurde beim Speichern der Datei. Der XML-Header der Datei enthielt dann auch den String „UTF-8“, wodurch der XML-Parser beim Laden versuchte, die Datei entsprechend zu interpretieren. Sobald jedoch ein Umlaut z.B. im Spielername auftauchte, führte das zu einer ungültigen UTF-8-Sequenz und das Laden brach an dieser Stelle mit einer entsprechenden Exception ab. Unter Linux trat das Problem sowohl mit UTF-8-Locales als auch mit ISO-8859-Locales nicht auf. Da der Grund für dieses Verhalten unbekannt ist, wurde das Problem nur umgangen, indem unter Windows „ISO-8859-15“ in den XML-Header geschrieben wird. Für die anderen geladenen XML-Dateien ist das Problem nicht relevant, weil diese nicht zur Laufzeit des Spiels bearbeitet werden.

7 Quellcodes

Die Quellcodes befinden sich nur deswegen in der Datei, damit man über die Referenzen im Text oder das PDF-Inhaltsverzeichnis direkt bestimmte Quelldateien anspringen und die eine oder andere Sache nachlesen kann.

Listing 1: Game.java

```
1  package name.panitz.java.ws0607;

   public interface Game {
4   void start();
   }
```

Listing 2: IPartikel.java

```
1  package de.rccc.java.witchcraft;

   /**
4   * Dieses Interface muss jede Klasse implementieren, die
    * von der PartikelVerwaltung als Partikel angesehen werden will
    */
7  public interface IPartikel {
    /**
     * Methode zur Darstellung
10   */
    public void zeichne(java.awt.Graphics g);

13  /**
     * Methode zum Aufaddieren eines Graviationsfaktors
     * (alle Partikel sinken nach unten)
16   */
    public void gravitationAdd(double g);

19  /**
     * Methode zum Multiplizieren mit einem Gravitationsfaktor
     * (alle Partikel beschleunigen nach unten)
22   */
    public void gravitationMult(double g);

25  /**
     * Methode um den Partikel einen Schritt weiterzubewegen
     */
28  public void bewege();

    /**
31   * Überprüft, ob der Partikel ausserhalb des Bildschirms ist
     *
     * @return true, wenn das Ojekt nicht mehr gebraucht wird
34   */
    public boolean ausserhalbBildschirm();
   }
```

Listing 3: Main.java

```
   package de.rccc.java.witchcraft;

3  import java.awt.*;
   import java.awt.event.*;
```

```
import java.util.*;
import javax.swing.*;
import name.panitz.java.ws0607.Game;

/**
 * Diese Klasse sorgt dafuer, dass ein neuer JFrame gezeichnet wird,
 * die Menueleiste zugefuegt und die TAnzeige (dh das eigentliche Spiel)
 * erzeugt, dargestellt und gestartet wird
 */
public class Main implements Game {
 /**
  * Die TAnzeige, die das eigentliche Spiel enthaelt
  */
 final protected TAnzeige anzeige;

 /**
  * Das Fenster, in dem wir spielen wollen
  */
 final protected JFrame fhauptFenster = new JFrame("Witchcraft");

 /**
  * HashMap der Menueeintraege
  */
 final protected Map<String, JMenuItem> fmenuItems =
  new HashMap<String, JMenuItem>();

 /**
  * Ausgrauen oder enablen eines bestimmen Menueintrages
  * @param item Beschreibung des menuItems, z.B. "spiel_neu"
  * @param enabled true fuer enabled, false fuer disablen
  */
 public void enableMenuItem(String item, boolean enabled) {
  fmenuItems.get(item).setEnabled(enabled);
 }

 /**
  * Aendert den Text eines MenuItems
  * @param item Beschreibung des menuItems, z.B. "optionen_musik"
  * @param neuerText Der neue zu setzende Text
  */
 public void aendereMenuItem(String item, String neuerText) {
  fmenuItems.get(item).setText(neuerText);
 }

 /**
  * Liefert das Fenster zurueck - damit man es schliessen
  * kann, wenn das Spiel als Applet laeuft (dort funktioniert
  * System.exit() nicht)
  */
 public JFrame getHauptFenster() {
  return fhauptFenster;
 }

 /**
  * Defaultkonstruktor - um die Aufgabenstellung zu befriedigen
  */
 public Main() {
  this(false);
 }
```

```
  /**
66 * Konstruktor.
   * Hier wird das Fenster erzeugt, gefuellt und gestartet
   * @param applet Laeuft das Spiel in einem Applet? Das
69 * beeinflusst z.B. ob es einen Menuepunkt "Datei - Beenden"
   * gibt oder nicht (System.exit() nicht moeglich in Applets)
   */
72 public Main(final boolean applet) {
   // als Haupt-Spielklasse registrieren
   TSharedObjects.setMain(this);
75
   anzeige = new TAnzeige(applet);
   JMenuItem tmp;
78
   // Tasten-Releaseevents werden nicht erfasst, wenn wir
   // auf dem Menu rumklicken, daher mal praeventiv alle
81 // Tasten loslassen - wir wollen nicht dass der Spieler
   // "weiterschlittert"
   final MouseMotionAdapter mma = new MouseMotionAdapter() {
84  public void mouseDragged(MouseEvent _) {
     anzeige.tastenLoslassen();
     anzeige.pause();
87  }

    public void mouseMoved(MouseEvent _) {
90   anzeige.tastenLoslassen();
     anzeige.pause();
    }
93 };

   tmp = new JMenuItem("Neu");
96 tmp.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent _) {
     new TOptionenFenster(applet);
99 }});
   tmp.addMouseMotionListener(mma);
   fmenuItems.put("spiel_neu", tmp);
102
   tmp = new JMenuItem("Laden");
   tmp.addActionListener(new ActionListener() {
105  public void actionPerformed(ActionEvent _) {
     TSavegame.savegameWaehlen();
   }});
108 tmp.addMouseMotionListener(mma);
   fmenuItems.put("spiel_laden", tmp);

111 tmp = new JMenuItem("Speichern");
   tmp.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent _) {
114  anzeige.speichere();
   }});
   tmp.addMouseMotionListener(mma);
117 fmenuItems.put("spiel_speichern", tmp);

   tmp = new JMenuItem(applet ? "Fenster␣schliessen" : "Beenden");
120 tmp.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent _) {
     TSharedObjects.endGame();
```

```
123     }});
        tmp.addMouseMotionListener(mma);
        fmenuItems.put("spiel_beenden", tmp);
126
        tmp = new JMenuItem("Hohe Details (schneller PC)");
        tmp.addActionListener(new ActionListener() {
129      public void actionPerformed(ActionEvent _) {
          anzeige.setHohesDetail(true);
          TSharedObjects.getPartikelVerwaltung().setHohesDetail(true);
132     }});
        tmp.addMouseMotionListener(mma);
        fmenuItems.put("optionen_hidetail", tmp);
135
        tmp = new JMenuItem("Niedrige Details (lahme Krücke)");
        tmp.addActionListener(new ActionListener() {
138      public void actionPerformed(ActionEvent _) {
          anzeige.setHohesDetail(false);
          TSharedObjects.getPartikelVerwaltung().setHohesDetail(false);
141     }});
        tmp.addMouseMotionListener(mma);
        fmenuItems.put("optionen_lodetail", tmp);
144
        tmp = new JMenuItem(TSharedObjects.getMusik() ? "Musik abschalten" :
         "Musik anschalten");
147     tmp.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent _) {
          TSharedObjects.setMusik(!TSharedObjects.getMusik());
150     }});
        tmp.addMouseMotionListener(mma);
        fmenuItems.put("optionen_musik", tmp);
153
        tmp = new JMenuItem("Anleitung");
        tmp.addActionListener(new ActionListener() {
156      public void actionPerformed(ActionEvent _) {
          anzeige.pauseDurchMenu();
          TMenuHTMLzeig ueb = new TMenuHTMLzeig("anleitung.html");
159     }});
        tmp.addMouseMotionListener(mma);
        fmenuItems.put("hilfe_anleitung", tmp);
162
        tmp = new JMenuItem("Über");
        tmp.addActionListener(new ActionListener() {
165      public void actionPerformed(ActionEvent _) {
          anzeige.pauseDurchMenu();
          TMenuHTMLzeig ueb = new TMenuHTMLzeig("ueber.html");
168     }});
        tmp.addMouseMotionListener(mma);
        fmenuItems.put("hilfe_ueber", tmp);
171
        final JMenuBar hauptMenue = new JMenuBar();
        final JMenu menue_spiel = new JMenu("Spiel");
174     final JMenu menue_optionen = new JMenu("Optionen");
        final JMenu menue_hilfe = new JMenu("Hilfe");

177     enableMenuItem("spiel_speichern", false);
        enableMenuItem("spiel_laden", false);
        menue_spiel.add(fmenuItems.get("spiel_neu"));
180     menue_spiel.addSeparator();
        menue_spiel.add(fmenuItems.get("spiel_laden"));
```

```
    menue_spiel.add(fmenuItems.get("spiel_speichern"));
    menue_spiel.addSeparator();
    menue_spiel.add(fmenuItems.get("spiel_beenden"));
    menue_spiel.addMouseMotionListener(mma);
    menue_optionen.add(fmenuItems.get("optionen_hidetail"));
    menue_optionen.add(fmenuItems.get("optionen_lodetail"));
    menue_optionen.addSeparator();
    menue_optionen.add(fmenuItems.get("optionen_musik"));
    menue_optionen.addMouseMotionListener(mma);
    menue_hilfe.add(fmenuItems.get("hilfe_anleitung"));
    menue_hilfe.add(fmenuItems.get("hilfe_ueber"));
    menue_hilfe.addMouseMotionListener(mma);
    hauptMenue.add(menue_spiel);
    hauptMenue.add(menue_optionen);
    hauptMenue.add(menue_hilfe);
    hauptMenue.addMouseMotionListener(mma);

    fhauptFenster.setJMenuBar(hauptMenue);
    fhauptFenster.setResizable(false);

    if (applet) {
     enableMenuItem("spiel_laden", false);
    } else {
     fhauptFenster.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     if (TSavegame.mindEinSavegameVorhanden()) {
      enableMenuItem("spiel_laden", true);
     }
    }

    fhauptFenster.add(anzeige);
    fhauptFenster.pack();
  }

  /**
   * Um das geforderte Interface implementieren zu koennen:
   * Macht das Spielfenster sichtbar und startet das Spiel im Panel
   */
  public void start() {
   fhauptFenster.setVisible(true);
   anzeige.start();
  }

  /**
   * Main-Funktion zum normalen Starten des Spiels
   */
  public static void main(String [] _) {
   Game g = new Main();
   g.start();
  }
}
```

Listing 4: SpielApplet.java

```
package de.rccc.java.witchcraft;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

```

```
/**
 * Klasse, die das Spiel in ein JApplet wrappt
 */
public class SpielApplet extends JApplet {
 /**
  * Konstruktor, der auch die gesamte funktionalitaet enthaelt
  * Dargestellt wird ein Hinweislabel und ein Knopf, der dann
  * das eigentliche Spiel in einem Neuen Fenster startet
  */
 public SpielApplet() {
  final name.panitz.java.ws0607.Game g = new Main(true);

  final JPanel panel = new JPanel();
  final JButton startButton = new JButton("Spiel Starten");
  startButton.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent _) {
    g.start();
   }
  });
  final JLabel label1 = new JLabel(
   "Damit das Spiel funktioniert, wird ein Extra Fenster benötigt.");
  final JLabel label2 = new JLabel(
   "Drücken Sie den Knopf, um das Spiel in einem Fenster zu öffnen");

  panel.add(label1);
  panel.add(label2);
  panel.add(startButton);
  add(panel);
 }
}
```

Listing 5: TAnimation.java

```
package de.rccc.java.witchcraft;

import java.awt.image.BufferedImage;
import java.util.*;
import javax.imageio.*;
import java.awt.geom.AffineTransform;
import java.awt.image.AffineTransformOp;

/**
 * TAnimation kapselt eine Liste von Bildern und eine
 * Animationsgeschwindigkeit in ein Objekt. Sie hat ausserdem
 * die Methode "rotiereFrame", die aufgerufen werden kann, wenn
 * das Objekt genau einen Frame enthaelt. Dieser wird dann in
 * mehrfach gedrehter Form als die Liste der Frames verwendet,
 * fuer Geschosse.
 */
public class TAnimation {
 /**
  * Anzahl der Phasen fuer ein rotierendes Geschoss
  */
 static private int rphasen = 32;

 /**
  * Liste der Animationsphasen
  */
 protected List<BufferedImage> fphasen = new ArrayList<BufferedImage>();
```

```
27
   /**
    * Der Zaehler der angibt, wie oft der aktuelle Animationsframe
30  * bereits wiederholt wurde
    */
   protected int faktuell = 0;
33
   /**
    * Der Zaehler, der angibt, welcher der Animationsframes der aktuell
36  * anzuzeigende ist
    */
   protected int fphase = 0;
39
   /**
    * Wie oft soll jeder Animationsframe wiederholt werden
42  */
   protected int fwiederhol;

45 /**
    * Konstruktor fuer ein einzelnes Bild
    *
48  * @param bildname Dateiname des Bildes
    */
   TAnimation(String bildname) {
51  fwiederhol = 0;
    fphasen.add(ladeBild(bildname));
   }
54
   /**
    * Konstruktor fuer mehrere Animationsphasen
57  *
    * @param bildmaske Dateinamen-maske der Phasenbilder, dieser muss
    * im Format sein: bild%%.png (genau ein Vorkommen von '%%', dieses
60  * wird spaeter durch die Nummern der Einzelbilder ersetzt (1-n))
    * @param anzahl Wie viele Einzelbilder sollen geladen werden. Ein Wert von
    * 3 laedt mit der Maske "bild%%.png" die Dateien "bild1.png", "bild2.png"
63  * und "bild3.png"
    * @param wiederhol Die Anzahl der Ticks, die jedes Einzelbild vor dem
    * Wechsel angezeigt werden soll
66  */
   TAnimation(String bildmaske, int anzahl, int wiederhol) {
    fwiederhol = wiederhol;
69  for (int i = 1; i <= anzahl; i++) {
     fphasen.add(ladeBild(bildmaske.replace("%%", "" + i)));
    }
72 }

   /**
75  * Kopierkonstruktor.
    * Beim Kopieren der TAnimation bekommt die Kopie eine Referenz
    * (keine Kopie) der Liste der Animationsframes.
78  * @param t Die zu kopierende TAnimation
    */
   TAnimation(TAnimation t) {
81  this.fwiederhol = t.fwiederhol;
    this.fphasen = t.fphasen;
    this.faktuell = fwiederhol > 0 ? TSharedObjects.rndInt(fwiederhol) : 0;
84  this.fphase = t.fphasen.size() > 0 ?
     TSharedObjects.rndInt(t.fphasen.size()) : 0;
```

```
  }
87
  /**
   * Liefert den Groessenvektor des ersten Animationsframes zurueck
90 * (es wird davon ausgegangen, dass alle Frames einer Animation
   * die selbe Groesse haben)
   */
93 public TVektor groesse() {
   return groesse(fphase);
  }
96
  /**
   * Liefert den Groessenvektor eines beliebigen Frames zurueck.
99 * Diese Methode wird bei Geschossen verwendet, da hier die Frames
   * unterschiedliche Groessen haben koennen.
   * @param frame Die Nummer des Frames, dessen Groesse geliefert
102 * werden soll
   */
  public TVektor groesse(int frame) {
105  return new TVektor(fphasen.get(frame).getWidth(null),
      fphasen.get(frame).getHeight(null));
  }
108
  /**
   * Liefert den naechsten anzuzeigenden Frame bei Animationen
111 * mit gleichmaessigem Tempo
   * @return Der Animationsframe
   */
114 public BufferedImage getFrame() {
   faktuell++;
   if (faktuell > fwiederhol) {
117  faktuell = 0;
     fphase++;
     if (fphase >= fphasen.size()) {
120   fphase = 0;
     }
   }
123  return fphasen.get(fphase);
  }

126 /**
   * Liefert einen der Frames bei Animationen, wo die Frames
   * fuer die verschiedenen Ausrichtungen eines Objektes stehen,
129 * abhaengig von der gewuenschten Ausrichtung
   *
   * @param ausrichtung Die Ausrichten des Objektes in RAD
132 * @return Die Drehphase
   */
  public BufferedImage getFrame(double ausrichtung) {
135  // Welches Frame wird gebraucht?
   int x = (int)(rphasen - (rphasen / (2 * Math.PI)) *
     (ausrichtung - 3 * Math.PI)) % rphasen;
138  return fphasen.get(x);
  }

141 /**
   * Laedt ein Bild aus einer Datei
   */
144 private BufferedImage ladeBild(String dateiname) {
```

```
    BufferedImage res = null;
    java.net.URL pfad =
147   this.getClass().getClassLoader().getResource("bilder/" + dateiname);
    try {
     res = ImageIO.read(pfad);
150 } catch (Exception e) {
     System.out.println("Konnte Bild " + dateiname +
      " nicht laden (Pfad: " + pfad + ")");
153  TSharedObjects.endGame();
    }

156 return res;
   }

159 /**
    * Diese Methode kann einmalig aufgerufen werden fuer TAnimations-Objekte,
    * die genau einen Frame haben.
162 * Sie dreht dann diesen Frame rphasen-1 mal (z.Zt. 31 mal),
    * hinterher hat das TAnimation-Objekt 32 Frames, die jeweils um 11.25°
    * (= 360/32) gedreht sind. Das macht Sinn fuer Geschosse, damit diese
165 * nicht bei jedem mal Zeichnen die teuere Transformations-Operation
    * durchfuehren muessen.
    */
168 public void rotiereFrame() {
    // Wir wollen nur Frames erzeugen bei Animationen mit genau einem Frame
    // Allerdings verwenden mehrere Waffen die selben Animations-Objekte,
171 // deswegen ignorieren wir einen erneuten Aufruf (kein Problem)

    if (fphasen.size() == 1) {
174  BufferedImage orig = fphasen.get(0);

     // Die ursprüngliche Dimension merken:
177  TVektor fdim = new TVektor(orig.getWidth(), orig.getHeight());

     int groesse = (int)fdim.x;
180  if (groesse < fdim.y) {
      groesse = (int)fdim.y;
     }
183
     // Das Originalbild durch ein quadratisches Bild
     // ersetzen, damit das Drehen klappt
186  BufferedImage neuOriginal = new BufferedImage(groesse, groesse,
      BufferedImage.TYPE_INT_ARGB_PRE);
     neuOriginal.createGraphics().drawImage(orig, null,
189   (groesse - orig.getWidth()) / 2, (groesse - orig.getHeight()) / 2);
     orig = neuOriginal;

192  double w = groesse / 2;
     double theta = (double)(Math.toRadians(360 / rphasen));
     int transformTyp = AffineTransformOp.TYPE_BICUBIC;
195  AffineTransformOp transformer = null;

     // Es muss jedesmal ein neues Transformationsobjekt erzeugt werden.
198  // Wuerden wir inkrementell drehen, d.h. den n-ten Frame basierend
     // auf dem (n-1)-ten, so wuerden die spaeteren Frames verwaschen
     // aussehen
201  for (int i = 1; i < rphasen; i++) {
      transformer = new AffineTransformOp(
       AffineTransform.getRotateInstance(theta * i, w, w), transformTyp);
```

```
204       BufferedImage neu = new BufferedImage(groesse, groesse,
           BufferedImage.TYPE_INT_ARGB_PRE);
          transformer.filter(orig, neu);
207       fphasen.add(neu);
         }

210      fwiederhol = 1;
       }
     }
213  }
```

Listing 6: TAnzeige.java

```
1  package de.rccc.java.witchcraft;

   import java.awt.*;
4  import java.awt.event.*;
   import javax.swing.*;
   import java.util.*;
7  import javax.xml.parsers.*;
   import org.w3c.dom.*;

10 /**
    * TAnzeige ist die eigentliche Hauptklasse des Spiels: Hier gibt es
    * Listen fuer bewegte Objekte, Objekte fuer Spieler, Fadenkreuz usw,
13  * sowie die Maus- und Tastaturlistener
    */
   class TAnzeige extends JPanel {
16  /**
     * Am LevelEnde soll das Bild noch so lange stehen bleiben
     */
19  protected final static int fLEVELSTEHEN = 100;

    /**
22   * Gibt an, ob das Spiel als Applet ausgefuehrt wird (beeinflusst
     * zB, ob Laden und Speichern moeglich sein soll
     */
25  protected boolean fapplet;

    /**
28   * Vektor, der die Dimension des Fenster enthaelt
     */
    protected TVektor fdim;
31
    /**
     * Liste der Geschosse, die auf der Spielflaeche rumfliegen
34   */
    protected java.util.List<TGeschoss> fgeschosse =
     new ArrayList<TGeschoss>();
37
    /**
     * Liste der Lebewesen (Spieler und Gegner)
40   */
    protected java.util.List<TLebewesen> flebewesen =
     new ArrayList<TLebewesen>();
43
    /**
     * Liste von Objekten ohne Spieleinfluss. Diese werden auch
46   * nicht bewegt, z.B. die Textobjekte, die die Sonderpunkte
```

```
     * anzeigen.
     */
49   protected java.util.List<TObjekt> fobjekte =
     new ArrayList<TObjekt>();

52   /**
     * Liste von Items (vom Spieler einsammelbare Gegenstaende)
     */
55   protected java.util.List<TItem> fitems =
     new ArrayList<TItem>();

58   /**
     * Der spielablaufsteuernde Timer
     */
61   protected javax.swing.Timer fzeitgeber;

     /**
64   * Der Timer, der fuer das Neuzeichnen des Fensters verantwortlich ist
     */
     protected javax.swing.Timer fupdater;
67
     /**
     * Der Spieler
70   */
     protected TSpieler fspieler = null;

73   /**
     * Das Fadenkreuz
     */
76   protected TBildObjekt ffadenkreuz = null;

     /**
79   * Unsichtbarer Cursor (damit das Fadenkreuz nicht von einem haesslichen
     * Pfeil ueberlagert wird)
     */
82   protected Cursor finvisiCursor = null;

     /**
85   * Das Objekt, das den Level darstellt
     */
     protected TLevel flevel = null;
88
     /**
     * Hat einen Wert zwischen 0 und 1, der angibt, wie sichtbar stark
91   * das Spielfeld ausgefaded ist (verdunkelt)
     */
     protected double ffader = 1.;
94
     /**
     * Gibt an, ob gerade gefaded werden soll: -1 einfaden, 0 nicht faden,
97   * 1 ausfaden
     */
     protected int ffade = 0;
100
     /**
     * Die Farbe, zu der hin gefaded werden soll
103  */
     protected Color ffadeColor = new Color(0, 0, 0);
```

```
106  /**
     * Ist hohes Grafikdetail eingestellt? (Betrifft Partikelsystem
     * und Hintergrund)
109  */
     protected boolean fhdetail = true;

112  /**
     * Einzeilige Textmeldung, die ueber allen anderen Sachen gezeichnet wird.
     * Fuer Meldungen wie "Spiel gespeichert." usw.
115  */
     protected TText fmeldung = null;

118  /**
     * Gibt an, ob bereits ein Spiel gestartet wurde (mittels Neu/Laden)
     */
121  protected boolean fspielGestartet = false;

     /**
124  * Gibt an, ob das Spiel gerade pausiert ist
     */
     protected boolean fpausiert = false;
127
     /**
     * Gibt an, ob das Level gerade gestartet wurde, und noch
130  * die Stats weiter angezeigt werden sollen (d.h., dieser
     * Wert darf nur true werden, wenn wir schon ein Level gespielt
     * haben)
133  */
     protected boolean flevelGestartet = false;

136  /**
     * Gibt den aktuellen Schwierigkeitsgrad an
     */
139  protected TSpieler.Schwierigkeitsgrade fschwierigkeit =
      TSpieler.Schwierigkeitsgrade.normal;

142  /**
     * Ist dieser Wert != 0, so "wackelt" das Bild entsprechend
     * (bei Explosionen)
145  */
     protected int fbeben = 0;

148  /**
     * Nachdem das Level vorbei ist, soll es noch etwas angezeigt werden.
     */
151  protected int flevelwarte = fLEVELSTEHEN;

     /**
154  * Getter fuer die Lebewesen-Liste
     * @return Liste der Lebewesen
     */
157  public java.util.List<TLebewesen> getLebewesen() {
      return flebewesen;
     }
160
     /**
     * Methode, die bestimmt, wie gross das Fenster dargestellt wird
163  * @return Dimension-Objekt, das die Fenstergroesse repraesentiert
     */
```

```
  public java.awt.Dimension getPreferredSize() {
166  return new java.awt.Dimension((int)fdim.x, (int)fdim.y);
  }

169  /**
   * Startet das Einfaden (aufhellen)
   */
172  public void fadeIn() {
   ffader = 1.;
   ffade = -1;
175  }

   /**
178  * Startet das Ausfaden (abdunkeln)
   */
   public void fadeOut() {
181  ffader = 0.;
   ffade = 1;
   }
184

   /**
   * Bringt den Alphawert fuer das Faden auf den neuesten Stand
187  * (faded ein oder aus, je nach momentaner Einstellung)
   */
   public void fadeUpdate() {
190  if (ffade != 0) {
    ffader += ((double)ffade / 10);
    if (ffader < 0) {
193   ffader = 0;
     ffade = 0;
    } else if (ffader > 1) {
196   ffader = 1;
     ffade = 0;
    }
199  }
   }

202  /**
   * Methode, die beim Zeichnen des Fensters aufgerufen wird.
   * Diese Methode steuert alle Sichtbarkeiten und ruft ggf.
205  * Zeichenmethoden von Unterobjekten auf
   * @param g Die Zeichenflaeche, auf der gezeichnet werden soll
   */
208  public void paintComponent(java.awt.Graphics g) {
   super.paintComponent(g);
   if (fhdetail) {
211   g.drawImage(TSharedObjects.getBild("bg").getFrame(), 0, 0, null);
   }

214  int beben = 0;
   if (fbeben > 0) {
    beben = TSharedObjects.rndInt(fbeben*2) - fbeben;
217   g.translate(beben, beben);
   }

220  flevel.zeichne(g);

   for (TLebewesen l: flebewesen) {
223   l.zeichne(g);
```

```
    }

    for (TGeschoss ge: fgeschosse) {
     ge.zeichne(g);
    }

    for (TItem i: fitems) {
     i.zeichne(g);
    }

    for (TObjekt o: fobjekte) {
     o.zeichne(g);
    }

    TSharedObjects.getPartikelVerwaltung().zeichne(g);

    if (beben != 0) {
     g.translate(-beben, -beben);
     fbeben--;
    }
    ffadenkreuz.zeichne(g);

    fadeUpdate();
    if (ffader > 0.001) {
     Graphics2D g2 = (Graphics2D)g;
     Composite originalComposite = g2.getComposite();
     g2.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,
      (float)ffader / 2));
     g2.setColor(ffadeColor);
     g2.fillRect(0, 0, (int)fdim.x, (int)fdim.y);
     g2.setComposite(originalComposite);
    }

    flevel.zeichneMeldung(g);
    fmeldung.zeichne(g);
    if ((flevelwarte < fLEVELSTEHEN) || flevelGestartet) {
     // Der Spieler kann dann noch abkratzen...
     if (fspieler != null) {
      fspieler.zeichneStats(g);
     }
    }
   }

   /**
    * Geschosse bewegen und auf Kollision abfragen
    */
   private void bewegeGeschosse() throws Exception {
    for (Iterator<TGeschoss> i = fgeschosse.iterator();i.hasNext();) {
     TGeschoss ge = i.next();
     ge.bewege();
     boolean weiter = true;
     boolean weg = false;
     for (Iterator<TLebewesen> j = flebewesen.iterator();
      j.hasNext() && weiter;) {

      TLebewesen l = j.next();
      if (ge.beruehrt(l)) {
       ge.ende(true);
       if (l.treffer(ge.getTrefferp())) {
```

```
283         l.ende(true, ge.getWaffe());
            j.remove();

286         // der Spieler hat ausgespielt
            if (fspieler == l) {
             setGameOver();
289          fspieler = null; // NACH der Meldung!
             TSound.play("GAMEOVER");
            }
292        }
           // Die Rail z.B. soll etwas stehen bleiben
           if (ge.getTtl() < 0) {
295         weg = true;
           }
           // Ein Einschlag reicht
298        weiter = false;
         }
        }
301
        if ((ge.ausserhalbBildschirm()) || ge.tot()) {
         if (weg) {
304       // Das Geschoss ist aufgeschlagen
          ge.ende(false);
         }
307      weg = true;
        }
        if (weg) {
310      i.remove();
        }
       }
313   }

      /**
316    * Zeigt die Gameover-Meldung und die aktuelle Punktzahl
       */
      private void setGameOver() {
319    flevel.setAktuelleMeldung("GAMEOVER");
       addObjekt(new TText(new TVektor(150, 100),
        "Erreichte Punkte: " + fspieler.getScore(),
322     new Font("Arial", Font.BOLD, 28)));
      }

325   /**
       * Die Methode, die fuer das Bewegen aller Objekte zustaendig ist
       */
328   public void bewege() throws Exception {
       flevel.weiterScrollen();

331    for (Iterator<TLebewesen> i = flebewesen.iterator(); i.hasNext();) {
        TLebewesen l = i.next();
        l.bewege();
334     // Wenn das Lebewesen außerhalb des Bildschirms ist, weg damit
        if (l.ausserhalbBildschirm()) {
         l.ende(false, null);
337      i.remove();
        }
       }
340
       bewegeGeschosse();
```

```
343   // Die Items durchiterieren..
      // Auf Kollission ueberpruefen nur, wenn der Spieler noch existiert
      for (Iterator<TItem> i = fitems.iterator(); i.hasNext();) {
346    TItem akt = i.next();
       akt.bewege();
       if (fspieler != null) {
349     if (fspieler.beruehrt(akt)) {
         fspieler.addItem(akt);
         i.remove();
352      akt.ende(true);
        }
       }
355    if ((akt.ausserhalbBildschirm())) {
        i.remove();
        akt.ende(false);
358    }
      }

361   TSharedObjects.getPartikelVerwaltung().update();

      // Level geschafft?
364   if ((fspieler != null) && (flebewesen.size() == 1) &&
        (fitems.size() == 0) && (flebewesen.get(0) == fspieler) &&
        (flevel.levelZuEnde())) {
367    // einmal reicht nicht: im alten Level die Meldung setzen
       // (vor dem Laden), und auch im neu geladenen level
       flevel.setAktuelleMeldung("LEVELPASSED");
370    flevelwarte--;
       if (flevelwarte <= 0) {
        naechsterLevel();
373    }
      }
     }
376

     /**
      * Laedt den naechsten Level und zeigt eine entsprechende Meldung
379   */
     private void naechsterLevel() {
      TLevel oldlevel = flevel;
382   reset();
      try {
       if (flevel == null) {
385     flevel = new TLevel(1);
       } else {
        flevel = new TLevel(flevel.getLevelNummer() + 1);
388    }
       // einmal reicht nicht: im alten Level die Meldung setzen
       // (vor dem Laden), und auch im neu geladenen Level,
391    // weil die Meldung ueber Levelgrenzen hinweg stehen soll
       flevel.setAktuelleMeldung("LEVELPASSED");
       flevelwarte = fLEVELSTEHEN;
394    flevelGestartet = true;
       pause("Mit␣Leertaste␣weiterspielen");
      } catch (Exception e) {
397    // wir haben keine Level mehr...
       // damit das nicht schlecht aussieht, muss jedes level
       // mit einer Bildschirmbreite Einheitssegmente enden
400    // Es könnten zwar auch andere Fehler auftreten, aber die
```

```
    // Reaktion darauf soll die Gleiche bleiben.
    System.out.println(e);
403   flevel = oldlevel;
    setGameOver();
    fspieler = null;
406  }
   }

409  /**
    * Liefert das Lebewesen, das sich unter dem Fadenkreuz befindet,
    * sofern vorhanden
412  * @param x Die X-Koordinate, die ueberprueft werden soll
    * @param y Die Y-Koordinate, die ueberprueft werden soll
    * @param nichtseite Das Lebewesen soll nicht dieser Seite angehören
415  * (z.b. Soll der Spieler sich nicht selbst anvisieren)
    * @return null oder das Lebewesen unter dem Fadenkreuz
    */
418  public TLebewesen lebewesenUnterMaus(int x, int y, int nichtseite) {
    TLebewesen ret = null;

421  for (TLebewesen l: flebewesen) {
     if ((l.getSeite() != nichtseite) &&
      (l.innerhalb(new TVektor(x, y)))) {
424
      ret = l;
     }
427  }

    return ret;
430  }

   /**
433  * Das Spiel soll pausiert werden, weil jemand eins der Zusatz-
    * Fenster geoeffnet hat (mit entsprechender Meldung)
    */
436  public void pauseDurchMenu() {
    pause(fspielGestartet ? "Spiel␣pausiert␣-␣bitte␣Info-Fenster␣" +
     "schliessen␣zum␣Weiterspielen" : "␣");
439  }

   /**
442  * Pausiert das Spiel, mit Default-Meldung
    */
   public void pause() {
445  pause(fspielGestartet ? "PAUSE␣-␣Leertaste␣drücken␣zum␣" +
     "Weiterspielen" : "␣");
   }
448
   /**
    * Pausiert das Spiel, mit konfigurierbarer Meldung
451  * @param m Die Meldung, die auf der abgedunkelten Spielflaeche
    * stehen soll
    */
454  public void pause(String m) {
    // Wenn der Spieler nicht mehr ist, gibt es auch keine Pause mehr
    if ((fspieler != null) && !fpausiert) {
457   fpausiert = true;
    fadeOut();
    fmeldung.setText(m);
```

```
460     fzeitgeber.stop();
      }
    }
463
    /**
     * Startet den Zeitgeber (neu) und loescht evtl vorhandene
466  * Pause-Meldungen
     */
    public void start() {
469  // Um ein Laggen am Anfang zu vermeiden, rufen wir
     // den Garbage-Collector von Hand auf
     System.gc();
472
     if ((fspieler != null) && flevelGestartet) {
      fspieler.statsRuecksetzen();
475  }

     flevelGestartet = false;
478  fpausiert = false;
     fmeldung.setText(null);
     fadeIn();
481
     if (fspielGestartet) {
      fzeitgeber.start();
484   fupdater.start();
     }
    }
487
    /**
     * Fuegt ein Objekt zur allgemeinen Objektliste zu
490  * @param o Das Objekt, das zur Objektliste zugefuegt werden soll
     */
    public void addObjekt(TObjekt o) {
493  fobjekte.add(o);
    }

496 /**
     * Fuegt ein Geschoss zur Geschossliste zu
     * @param g Das Geschoss, das zur Geschossliste zugefuegt werden soll
499  */
    public void addGeschoss(TGeschoss g) {
     fgeschosse.add(g);
502 }

    /**
505  * Fuegt ein Lebewesen (Spieler oder Feind) zur Lebewesenliste zu
     * @param l Das Lebewesen, das zur Lebewesenliste zugfuegt werden soll
     */
508 public void addLebewesen(TLebewesen l) {
     flebewesen.add(l);
    }
511
    /**
     * Fuegt ein Item (vom Spieler einsammelbares Objekt) zur Itemliste zu
514  * @param i Das Item, das zur Itemliste zugefuegt werden soll
     */
    public void addItem(TItem i) {
517  fitems.add(i);
    }
```

```
520  /**
     * Wird aufgerufen, wenn die Maus bewegt wurde, und dieses
     * an den Spieler wietergegeben werden muss.
523  * @param x X-Koodiante der Maus
     * @param y Y-Koodiante der Maus
     */
526  public void aktionmouseMoved(int x, int y) {
     if (fspieler != null) {
      fspieler.zielgeaendert(lebewesenUnterMaus(x, y,
529    fspieler.getSeite()), x, y);
     }
    }
532
    /**
     * Setzten den Wert von fbeben und loest damit ein Wackeln
535  * des Bildschirms aus
     * @param beben das Beben
     */
538  public void setBeben(int beben) {
     fbeben = beben;
    }
541
    /**
     * Versucht, das Spiel zu speichern, und gibt eine Erfolgsmeldung
544  * aus, sofern angebracht
     */
    public void speichere() {
547  if (fspieler != null) {
      try {
       TSavegame.speichere(fspieler, flevel);
550    fmeldung.setTimeoutText("Spiel gespeichert.");
       TSharedObjects.getMain().enableMenuItem("spiel_laden", true);
      } catch (Exception e) {
553    fmeldung.setTimeoutText("Spiel speichern fehlgeschlagen!");
       e.printStackTrace();
      }
556  }
    }

559 /**
     * Bereitet alles auf das Laden eines Spieles vor, und laedt
     * anschliessend das Spiel
562  */
    public void lade(int spielnummer) {
     try {
565   fspielGestartet = true;
      pause();
      fupdater.stop();
568   reset();
      fspieler.setScore(0);
      fspieler.leereWaffen();
571   fspieler.statsRuecksetzen();
      flevel = TSavegame.lade(spielnummer, fspieler);
      flevelwarte = fLEVELSTEHEN;
574   fupdater.start();
      fmeldung.setText("Spiel geladen - Leertaste drücken zum " +
       "Weiterspielen");
577  } catch (Exception e) {
```

```
    fmeldung.setTimeoutText("Spiel laden fehlgeschlagen!");
    e.printStackTrace();
   }
  }

  /**
   * Setzt alles (Geschosse, Gegner, etc) zurueck, damit ein
   * neues oder geladenes Spiel sauber anfaengt
   */
  protected void reset() {
   setCursor(finvisiCursor);
   if (fspieler == null) {
    fspieler = new TSpieler(fschwierigkeit);
   } else {
    fspieler.setSchwierigkeit(fschwierigkeit);
    fspieler.leereWaffenListe();
   }

   TSharedObjects.getPartikelVerwaltung().reset();
   for(TLebewesen lw: flebewesen) {
    lw.ende(false, null);
   }
   flebewesen.clear();
   fgeschosse.clear();
   fitems.clear();
   fobjekte.clear();
   flebewesen.add(fspieler);
   fspieler.setKoord(new TVektor(5, 200));
   if (!fapplet) {
    TSharedObjects.getMain().enableMenuItem("spiel_speichern", true);
   }
   TWaffe.resetStatistik();
  }

  /**
   * Wird aufgerufen, wenn der Spieler im Menue "Spiel - Neu" auswaehlt
   * und sich im Optionen-Dialog durchgeklickt hat
   * @param name Der Name des Spielers
   * @param schwierigkeit Der Schwierigkeitsgrad
   */
  public void neuesSpiel(String name,
   TSpieler.Schwierigkeitsgrade schwierigkeit) {

   pause();
   fschwierigkeit = schwierigkeit;
   reset();
   try {
    flevel = new TLevel(1);
    flevelwarte = fLEVELSTEHEN;
   } catch (Exception e) {
    fmeldung.setTimeoutText("Level laden fehlgeschlagen!");
    e.printStackTrace();
   }
   fspielGestartet = true;
   fspieler.setName(name);
   flevel.deaktiviereMeldung();
   start();
  }
```

```
637  /**
     * Setzt das Grafikdetail und gibt eine Meldung aus ueber
     * die neue Einstellung
640  * @param d true fuer hohes Detail, false fuer niedriges Detail
     */
    public void setHohesDetail(boolean d) {
643  String meldung;

     fhdetail = d;
646  if (d) {
      meldung = "Hohes Detail ausgewählt.";
     } else {
649   meldung = "Niedriges Detail ausgewählt.";
     }

652  // nur melden wenn wir nicht grade pausieren
     if (!fspielGestartet || fzeitgeber.isRunning()) {
      fmeldung.setTimeoutText(meldung);
655  }
    }

658 /**
     * Setzt einen Info-Text
     * @param text Der zu setzende Text
661  */
    public void setMeldung(String text) {
     fmeldung.setTimeoutText(text);
664 }

    /**
667  * Bugfix: Wenn man eine Taste gedrueckt haelt und dann
     * auf das Menu klickt, "rastet" die Taste ein, dh ein
     * ReleaseEvent wird nicht mehr erfasst; daher rufen
670  * alle Menukomponenten diese Methode bei mouseEntered
     * Event auf
     */
673 public void tastenLoslassen() {
     if (fspieler != null) {
      fspieler.bewegen(TSpieler.Bewegung.rechts, false);
676   fspieler.bewegen(TSpieler.Bewegung.links, false);
      fspieler.bewegen(TSpieler.Bewegung.unten, false);
      fspieler.bewegen(TSpieler.Bewegung.oben, false);
679  }
    }

682 /**
     * Startet oder stoppt den Timer, der fuer das Neuzeichnen
     * des Fensters verantwortlich ist
685  * @param update true fuer Anschalten, false fuer Abschalten
     */
    public void updateEnable(boolean update) {
688  if (update) {
      fupdater.start();
     } else {
691   fupdater.stop();
     }
    }
694

    /**
```

```
     * Konstruktor.
697  * Dieser laedt Ressourcen (Grafiken, Sounds, Config), erzeugt
     * ein Levelobjekt (das seinerseits eine Leveldatei laedt) und erzeugt
     * Maus- und Tastaturlistener, deren Methoden den Spielablauf steuern
700  * @param applet Ist das Spiel im Applet gestartet (true) oder als
     * normale Anwendung (false)
     */
703  public TAnzeige(boolean applet) {
     // generelle Einstellungen
     fapplet = applet;
706  fdim = new TVektor(TSharedObjects.FENSTER_BREITE,
      TSharedObjects.FENSTER_HOEHE);
     fmeldung = new TText(new TVektor(40, 20), null,
709   new Font("Arial", Font.BOLD, 13), 10);

     // unsichtbaren Cursor generieren. Den setzen wir aber erst, wenns
712  // losgeht, weil es sonst verwirrend ist, wenn man keinen auf dem
     // Titelscreen sieht
     finvisiCursor = java.awt.Toolkit.getDefaultToolkit().createCustomCursor(
715   new java.awt.image.BufferedImage(1, 1,
      java.awt.image.BufferedImage.TYPE_4BYTE_ABGR),
      new java.awt.Point(0, 0), "NOCURSOR");
718
     // wir registrieren uns als Chef
     TSharedObjects.setAnzeige(this);
721
     // Spieltimer erzeugen
     fzeitgeber = new javax.swing.Timer(45, new ActionListener() {
724   public void actionPerformed(ActionEvent evt) {
       try {
        bewege();
727    } catch (Exception e) {
        System.out.println("Fehler im Spiel: " + e);
        e.printStackTrace();
730     System.out.println("Beende");
        TSharedObjects.endGame();
       }
733   }
     });

736  // Laden der Ressourcen
     try {
      TConfig.ladeRessourcen();
739   // Einige Sachen laden, damit die "Engine" läuft
      // Die werden sowieso wieder weggeschmissen
      fspieler = new TSpieler(
742    TSpieler.Schwierigkeitsgrade.normal);
      flevel = new TLevel(1);
      flevel.setAktuelleMeldung("TITEL");
745   //TSound.play("STARTSOUND");
     } catch(Exception e) {
      e.printStackTrace();
748   TSharedObjects.endGame();
     }

751  // Fadenkreuz erzeugen
     ffadenkreuz = new TBildObjekt(new TVektor(50, 50), null,
      new TVektor(0, 0), "FADENKREUZ");
754
```

```
    // Partikelverwaltung erzeugen und registrieren
    TSharedObjects.setPartikelVerwaltung(new TPartikelVerwaltung());
757 TSharedObjects.getPartikelVerwaltung().setHohesDetail(fhdetail);

    // Fenstereinstellungen
760 setFocusable(true);
    setBackground(new Color(70, 70, 70));

763 // Fenster-neu-zeichnen-Timer
    fupdater = new javax.swing.Timer(40, new ActionListener() {
     public void actionPerformed(ActionEvent evt) {
766   repaint();
     }
    });
769
    // Mouse-Listener
    addMouseMotionListener(new MouseMotionListener() {
772  public void mouseMoved(MouseEvent e) {
      ffadenkreuz.setKoord(new TVektor(e.getX()-23, e.getY()-23));
      aktionmouseMoved(e.getX(),e.getY());
775  }

     public void mouseDragged(MouseEvent e) {
778   ffadenkreuz.setKoord(new TVektor(e.getX()-23, e.getY()-23));
      aktionmouseMoved(e.getX(),e.getY());
     }
781 });
    addMouseListener(new MouseAdapter() {
     public void mouseClicked(MouseEvent e) {}
784
     public void mousePressed(MouseEvent e) {
      if (fspieler != null) {
787    if (e.getButton() == MouseEvent.BUTTON1) {
        fspieler.clickaktiv();
       }
790    if (e.getButton() == MouseEvent.BUTTON3) {
        fspieler.nextWaffe();
       }
793   }
     }

796  public void mouseReleased(MouseEvent e) {
      if (fspieler != null) {
       if (e.getButton() == MouseEvent.BUTTON1) {
799     fspieler.clicknichtaktiv();
       }
      }
802  }

     // Bugfix: Wenn man mit dem Cursor das Fenster betritt
805  // oder verlaesst, soll der Spieler anhalten - die Tasten
     // sollen "losgelassen" werden
     public void mouseEntered(MouseEvent e) {
808   tastenLoslassen();
     }

811  public void mouseExited(MouseEvent e) {
      tastenLoslassen();
     }
```

```
814     });

        // Keylistener
817     addKeyListener(new KeyAdapter() {
         public void keyPressed(KeyEvent e) {
          int kc = e.getKeyCode();
820
          // Nur wenn ein Spieler da ist, kann der Spieler reagieren
          if ((fspieler != null) && fspielGestartet) {
823        if ((kc == KeyEvent.VK_RIGHT) ||
            (kc == KeyEvent.VK_D)) {
           fspieler.bewegen(TSpieler.Bewegung.rechts, true);
826        }

           if ((kc == KeyEvent.VK_LEFT) ||
829            (kc == KeyEvent.VK_A)) {
           fspieler.bewegen(TSpieler.Bewegung.links, true);
           }
832
           if ((kc == KeyEvent.VK_UP) ||
            (kc == KeyEvent.VK_W)) {
835        fspieler.bewegen(TSpieler.Bewegung.oben, true);
           }

838        if ((kc == KeyEvent.VK_DOWN) ||
            (kc == KeyEvent.VK_S)) {
           fspieler.bewegen(TSpieler.Bewegung.unten, true);
841        }

           if (kc == KeyEvent.VK_N) {
844         fspieler.nextWaffe();
           }
          }
847     }

        public void keyReleased(KeyEvent e) {
850      int kc = e.getKeyCode();

         if (kc == KeyEvent.VK_SPACE) {
853       if (fspielGestartet) {
           if (fzeitgeber.isRunning()) {
            pause();
856        } else {
           flevel.deaktiviereMeldung();
           start();
859        }
          }
         }
862
         // Nur wenn ein Spieler da ist, kann der Spieler reagieren
         if ((fspieler != null) && fspielGestartet) {
865
          if ((kc == KeyEvent.VK_RIGHT) ||
           (kc == KeyEvent.VK_D)) {
868        fspieler.bewegen(TSpieler.Bewegung.rechts, false);
          }

871       if ((kc == KeyEvent.VK_LEFT) ||
           (kc == KeyEvent.VK_A)) {
```

```
        fspieler.bewegen(TSpieler.Bewegung.links, false);
      }

      if ((kc == KeyEvent.VK_UP) ||
        (kc == KeyEvent.VK_W)) {
        fspieler.bewegen(TSpieler.Bewegung.oben, false);
      }

      if ((kc == KeyEvent.VK_DOWN) ||
        (kc == KeyEvent.VK_S)) {
        fspieler.bewegen(TSpieler.Bewegung.unten, false);
      }
     }
    }
   });

   System.out.println("Temporäre Daten löschen...");
   // Garbage-Collector aufrufen, dadurch verhindern wir
   // das Laggen am Anfang
   System.gc();
  }
}
```

Listing 7: TBildObjekt.java

```
package de.rccc.java.witchcraft;

import java.util.Map;
import java.awt.image.*;

/**
 * Ein Spielobjekt, das durch ein Bild oder eine Animation
 * dargestellt wird (TAnimation-Objekt als Darstellung)
 */
public class TBildObjekt extends TObjekt implements IPartikel {
 /**
  * Darstellung des Objekts
  */
 protected TAnimation fbild;

 /**
  * Konstruktor
  *
  * @param koord Der Koordinatenvektor
  * @param dim Der Groessenvektor
  * @param geschw Der Geschwindigkeitsvektor
  * @param darstellung Darstellung des Objekts
  */
 TBildObjekt(TVektor koord, TVektor dim, TVektor geschw,
  String darstellung) {

  super(koord, dim, geschw);

  this.fbild = TSharedObjects.getNewBild(darstellung);
  if ((dim == null) && (fbild != null)) {
   this.fdim = fbild.groesse();
   if (fkoord != null) {
    this.fru = fkoord.newAdd(fdim);
   }
```

```
35    } else {
       this.fdim = dim;
      }
38   }

     /**
41    * Zeichnet das Bild an seinen Koordinaten auf der Zeichenflaeche g
      *
      * @param g Die Zeichenflaeche, auf der das Objekt gezeichnet werden soll
44    */
     public void zeichne(java.awt.Graphics g) {
      g.drawImage(fbild.getFrame(), (int)fkoord.x, (int)fkoord.y, null);
47   }
}
```

Listing 8: TConfig.java

```
package de.rccc.java.witchcraft;
2
import javax.xml.parsers.*;
import org.w3c.dom.*;
5 import java.util.*;

/**
8 * Klasse, die fuer das Laden von Ressourcen (Sound,
 * Bilder, Waffen) aus der Haupt-XML-Konfigurationsdatei
 * zustaendig ist
11 */
public class TConfig {

14 /**
   * Lädt die Waffenkonfiguration
   * @param n Aus diesem Node sollen die Waffen geladen werden
17 */
  private static void ladeWaffen(Node n) throws Exception {
   // in die Bilder eintauchen
20 NodeList nds = n.getChildNodes();
   // wie aktuell ;)
   NodeList a;
23 String id;
   TWaffe w;
   System.out.println("Lade Waffen");
26 for (int i = 0; i < nds.getLength(); i++) {
    if (nds.item(i) instanceof Element) {
     a = nds.item(i).getChildNodes();
29   if (nds.item(i).getNodeName()!="waffe") {
      throw new Exception("Das Element " + a +
       " hat hier Nichts zu suchen!");
32   } else {
      id = null;
      for (int j = 0; j < a.getLength(); j++) {
35     if (a.item(j).getNodeName() == "ID") {
        id = a.item(j).getTextContent();
       }
38    }
      if (id == null) {
       throw new Exception("Fehlendes ID Tag");
41    }
      System.out.println(" Lade: " + id);
```

```
      w = TWaffe.WaffeAusNode(nds.item(i));
      if (w == null) {
       throw new Exception("Fehler beim Laden der Waffe.");
      }
      TWaffe.addWaffe(id,w);
     }
    }
   }
  }

  /**
   * Läd die Sounds aus der übergebenen XML Liste
   * @param n Aus diesem Node sollen die Sounds geladen werden
   */
  private static void ladeSounds(Node n) throws Exception {
   System.out.println("Lese Sounds ein.. ");
   String id;
   String datei;
   boolean musik;
   // wie aktuell ;)
   NodeList a;
   // in die Bilder eintauchen
   NodeList nds = n.getChildNodes();

   for (int i = 0; i < nds.getLength(); i++) {
    if (nds.item(i) instanceof Element) {
     a = nds.item(i).getChildNodes();
     id = null;
     datei = null;
     musik = false;

     for (int j = 0; j < a.getLength(); j++) {
      if (a.item(j).getNodeName() == "id") {
       id = a.item(j).getTextContent();
      } else if (a.item(j).getNodeName() == "datei") {
       datei = a.item(j).getTextContent();
      } else if (a.item(j).getNodeName() == "musik") {
       // Alleine das vorhandensein reicht aus
       musik = true;
      }
     }

     if (datei == null || id == null) {
      throw new Exception("Fehler beim Einlesen von witchcraft.xml: " +
        "Fehlendes Element");
     }

     System.out.println(" Lade: " + id + "/" + datei);
     TSound.addSound(id, datei, musik);
    }
   }
  }

  /**
   * Lädt die Bilder aus der Übergebenen XML Liste
   *
   * @param n Der Node-Tree mit den bildern (der Punkt
   * mit "<bildern></bildern>")
   */
```

```
  private static void ladeBilder(Node n) throws Exception {
   System.out.println("Lese Bilder ein.. ");
104 String id;
   String datei;
   int anzahl;
107 int wiederhol;
   // wie aktuell ;)
   NodeList a;
110 // in die Bilder eintauchen
   NodeList nds = n.getChildNodes();

113 for (int i = 0; i < nds.getLength(); i++) {
    if (nds.item(i) instanceof Element) {
     a = nds.item(i).getChildNodes();
116  id = null;
     datei = null;
     anzahl = -1;
119  wiederhol = -1;

     for (int j = 0; j < a.getLength(); j++) {
122   if (a.item(j).getNodeName() == "ID") {
       id = a.item(j).getTextContent();
      } else if (a.item(j).getNodeName() == "datei") {
125   datei = a.item(j).getTextContent();
      } else if (a.item(j).getNodeName() == "anzahl") {
       anzahl = Integer.parseInt(a.item(j).getTextContent());
128   } else if (a.item(j).getNodeName() == "wiederhol") {
       wiederhol = Integer.parseInt(a.item(j).getTextContent());
      }
131  }

     if (datei == null || id == null) {
134   throw new Exception("Fehler beim Einlesen von witchcraft.xml: " +
        "Fehlendes Element");
     }
137
     System.out.println(" Lade: " + id + "/" + datei);

140  if ((anzahl < 0) && (wiederhol < 0)) {
      TSharedObjects.addBild(id, new TAnimation(datei));
     } else {
143   if ((anzahl < 0) || (wiederhol < 0)) {
       throw new Exception("Fehler beim Einlesen von witchcraft.xml: " +
         "anzahl und wiederhol müssen zusammen angegeben werden.");
146   } else {
       TSharedObjects.addBild(id, new TAnimation(datei,anzahl,wiederhol));
      }
149  }
    }
   }
  }
152 }

  /**
155  * Haupt-Methode, die das Laden der Ressourcen startet und steuert
   */
  public static void ladeRessourcen() throws Exception {
158  java.net.URL pfad = TConfig.class.getResource("/config/witchcraft.xml");
   if (pfad == null) {
    throw new Exception("/config/witchcraft.xml nicht gefunden");
```

```
161     }

        Document config = DocumentBuilderFactory
164             .newInstance().newDocumentBuilder()
                .parse(pfad.toURI().toString());
        NodeList nds = config.getChildNodes();
167
        // Wir erwarten 1 Hauptelement, das "witchcraft" sein soll
        if ((nds.getLength() != 1) ||
170      (nds.item(0).getNodeName() != "witchcraft")) {
         throw new Exception("Fehler: config-xml fehlerhaft formatiert");
        } else {
173      // in witchcraft eintauchen
         nds = nds.item(0).getChildNodes();
         for (int i = 0; i < nds.getLength(); i++) {
176       if (nds.item(i).getNodeName() == "bilder") {
           ladeBilder(nds.item(i));
          } if (nds.item(i).getNodeName() == "waffen") {
179        ladeWaffen(nds.item(i));
          } if (nds.item(i).getNodeName() == "sounds") {
           ladeSounds(nds.item(i));
182       }
         }
        }
185   }
}
```

Listing 9: TFeind.java

```
package de.rccc.java.witchcraft;
2
/**
 * Ein Feind-Objekt.
5 * Ein Feind ist eine autonome bewegende und feuernde Einheit,
 * könnte aber auch eine befreundete Einheit sein.
 * Nutzt TFeindDef.
8 */
public class TFeind extends TLebewesen {
 /**
11  * Die Feinddefiniton, auf dem dieser Feind aufbaut
  */
 protected TFeindDef ffeinddef = null;
14
 /**
  * Die Waffe, die dieser Feind verwendet.
17  * (Abkürzung, um einen Zugiff zu sparen)
  */
 protected TWaffe fwaffe = null;
20
 /**
  * Das Item, das dieser Feind fallenlässt
23  */
 protected TItem fitem;

26 /**
  * Das Bewegungsmuster, dem der Feind folgt
  */
29 protected int fmuster = -1;
```

```
  /**
32 * Der Parameter zu dem Bewegungsmuster: Berechnungsvariable
   */
  protected double fmusterparamberech = 0;
35
  /**
   * Der Parameter zu dem Bewegungsmuster: Laufvariable
38 */
  protected double fmusterparamlauf = 0;

41 /**
   * Liefert den "Bauplan" dieses Feindes
   */
44 public TFeindDef getFeindDef() {
   return ffeinddef;
  }
47
  /**
   * Konstruktor.
50 * Feinde koennen sich nach einem bestimmten Bewegungsmuster
   * fortbegegen, das die Flugbahn und die Geschwindigkeit bestimmt.
   * Mögliche Muster:
53 *                              langsam   schnell   extraschnell
   * geradeaus                      1         2          19
   * sinus flach                    3         5          20
56 * sinus hoch                     4         6          21
   * hoch langsam                   7         8
   * hoch schnell                   9        10
59 * runter langsam                11        12
   * runter schnell                13        14
   * zickzack, hoch                15        16
62 * zickzack, runter              17        18
   * reinkommend, dann hoch, runter 22        23
   * Kreis                        101
65 *
   * @param feinddef Die Feinddefinition (der Bauplan) fuer diesen Feind
   * @param startpunkt der Startpunkt des Feindes
68 * @param item Welches Item lässt dieser Feind nach der Zerstörung
   * liegen? (TItem/null)
   */
71 TFeind(TFeindDef feinddef, int startpunkt, int seite, TItem item) {

   super(null, null, new TVektor(), feinddef.getDarstellung(),
74  feinddef.getLebenspunkte(), seite);
   fwaffe = feinddef.getWaffe();

77 ffeinddef = feinddef;
   fitem = item;

80 // damit nicht alle sofort schiessen, wenn die auf den Bildschirm
   // erscheinen
   frof = TSharedObjects.rndInt(fwaffe.getRof());
83
   if ((startpunkt >= 0) && (startpunkt <= 40)) {
    this.fkoord = new TVektor(630, 10*startpunkt);
86 // Spezialmuster
   } else if (startpunkt == -2) {
    // Das UFO fliegt weg
89  this.fkoord = new TVektor(400, 50);
```

```
    } else {
     System.out.println("Fehler: Startpunkt " + startpunkt +
92    " unbekannt");
     this.fkoord = new TVektor(100, 100);
    }
95
    // langsam geradeaus
    fmuster = feinddef.getMuster();
98  if (fmuster == 1) {
     this.fgeschw.set(-2, 0);
    } else if (fmuster == 2) {
101  this.fgeschw.set(-5, 0);
    } else if (fmuster == 3) {
     this.fgeschw.set(-2, 0);
104  fmusterparamberech = 3;
    } else if (fmuster == 4) {
     this.fgeschw.set(-2, 0);
107  fmusterparamberech = 8;
     fmuster = 3; // gleiche berechnungsroutine
    } else if (fmuster == 5) {
110  this.fgeschw.set(-5, 0);
     fmusterparamberech = 3;
     fmuster = 3; // gleiche berechnungsroutine
113 } else if (fmuster == 6) {
     this.fgeschw.set(-5, 0);
     fmusterparamberech = 3;
116  fmuster = 3; // gleiche berechnungsroutine
    } else if (fmuster == 7) {
     this.fgeschw.set(-2, -1);
119 } else if (fmuster == 8) {
     this.fgeschw.set(-5, -1);
    } else if (fmuster == 9) {
122  this.fgeschw.set(-2, -3);
    } else if (fmuster == 10) {
     this.fgeschw.set(-5, -3);
125 } else if (fmuster == 11) {
     this.fgeschw.set(-2, 1);
    } else if (fmuster == 12) {
128  this.fgeschw.set(-5, 1);
    } else if (fmuster == 13) {
     this.fgeschw.set(-2, 3);
131 } else if (fmuster == 14) {
     this.fgeschw.set(-5, 3);
    } else if (fmuster == 15) {
134  this.fgeschw.set(-2, -3);
    } else if (fmuster == 16) {
     this.fgeschw.set(-5, -3);
137  fmuster = 15; // gleiche routine
    } else if (fmuster == 17) {
     this.fgeschw.set(-2, 3);
140  fmuster = 15; // gleiche routine
    } else if (fmuster == 18) {
     this.fgeschw.set(-5, 3);
143  fmuster = 15; // gleiche routine
    } else if (fmuster == 19) {
     this.fgeschw.set(-8, 0);
146 } else if (fmuster == 20) {
     this.fgeschw.set(-8, 0);
     fmusterparamberech = 3;
```

```
149   } else if (fmuster == 21) {
       this.fgeschw.set(-8, 0);
       fmusterparamberech = 8;
152    fmuster = 3;
      } else if (fmuster == 22) {
       this.fgeschw.set(-2, 0);
155   } else if (fmuster == 23) {
       this.fgeschw.set(-5, 0);
       fmuster = 22;
158   } else if (fmuster == 101) {
       // zuerst geradeaus, dann im Kreis
       this.fgeschw.set(-8,0);
161    fmusterparamlauf = -50;
      } else if (fmuster == -2) {
       // Das UFO fliegt weg
164    this.fgeschw = new TVektor(8, -1);
      } else {
       System.out.println("Fehler: Muster " + fmuster + " unbekannt");
167    this.fgeschw = new TVektor(0, 0);
      }
      fru = fkoord.newAdd(fdim);
170  }

     /**
173   * Bewegt das Objekt einen Schritt, abhaengig von seiner Geschwindigkeit
      * Außerdem wird hier die Waffe abgefeuert
      */
176  public void bewege(){
      frof -= 1;
      TAnzeige a = TSharedObjects.getAnzeige();
179
      // Ballern, wenn waffe bereit, und wenn Spieler noch existens
      if ((frof <= 0) && (a.fspieler != null)) {
182    frof = fwaffe.getRof();
       TVektor waffenausgang = ffeinddef.getWaffenausgang().newAdd(fkoord);

185    // Zielpunkt1: Mitte des Spielers
       TVektor zielp1 = new TVektor(a.fspieler.fdim);
       zielp1.mult(0.5);
188    zielp1.add(a.fspieler.fkoord);
       // Da wir ja einen Vektor brauchen, der auf das Ziel zeigt:
       // gleiche, wie der Auswurfpunkt!
191    zielp1.sub(waffenausgang);
       // Und jetzt dem Ding eine Geschwindigkeit geben.
       // Sonst ist der sofort da
194    zielp1.setlaenge(fwaffe.getGeschw());

       if (fwaffe.getWaffe() == TWaffe.Waffen.Rakete) {
197     TRakete rak = new TRakete(waffenausgang, null,
          new TVektor(fgeschw.x,fgeschw.x), fseite,
          fwaffe, 0, null, TVektor.genormt.winkel(zielp1),
200       fwaffe.getGeschw());
        a.addGeschoss(rak);
       } else if (fwaffe.getWaffe() == TWaffe.Waffen.Dumbfire) {
203     a.addGeschoss(new TGeschoss(waffenausgang, null,
         zielp1, fseite, fwaffe));
       } else if (fwaffe.getWaffe() == TWaffe.Waffen.Rail) {
206     // Das ist etwas unfair. Daher random..
        TVektor ziel = a.fspieler.getMitte();
```

```
      ziel.add(new TVektor(TSharedObjects.rndInt(200)-100,
        TSharedObjects.rndInt(200)-100));
      a.addGeschoss(new TRail(waffenausgang, a.fspieler, ziel,
        fseite, fwaffe));
    }
   }

   // Bei bestimmten Mustern jetzt die Bewegung aktualisieren
   if (fmuster == 3) {
    fmusterparamlauf += 0.1;
    fgeschw.y = (Math.sin(fmusterparamlauf) * fmusterparamberech);
   } else if (fmuster == 15) {
    fmusterparamlauf += 1;
    if (fmusterparamlauf > 40) {
     fmusterparamlauf = 0;
     fgeschw.y *= -1;
    }
   } else if (fmuster == 22) {
    if (fgeschw.y == 0) {
     // Fährt noch rein
     if (fkoord.x < (TSharedObjects.FENSTER_BREITE-fdim.x)) {
      // Ist weit genug drinnen
      fgeschw.swap();
     }
    } else {
     // Hoch / runter
     if ((fkoord.y < 0) && (fgeschw.y < 0)) {
      fgeschw.y *= -1;
     } else if ((fkoord.y > TSharedObjects.FENSTER_HOEHE-fdim.y) && (fgeschw.y > 0)) {
      fgeschw.y *= -1;
     }
    }
   } else if (fmuster == 101) {
    if (fmusterparamlauf >= 0) {
     // Ist weit genug drinnen. Jetzt kreis
     fmusterparamlauf  += 0.02;
     fgeschw.set(Math.cos(fmusterparamlauf)*4,-Math.sin(fmusterparamlauf)*4);
    } else {
     fmusterparamlauf++;
     if (fmusterparamlauf >= 0) {
      fmusterparamlauf = Math.PI;
     }
    }
   }

   super.bewege();
  }

  /**
   * Wird aufgerufen, wenn das Objekt gelöscht (aus den Listen) wird.
   * So kann jedes erbende Objekt ein letztes Röcheln abgeben.
   *
   * @param tot True, wenn das Objekt wirklich stirbt.
   * False, wenn das Objekt nur "getötet" wird, weil es
   * außerhalb des Bildschirms ist
   * @param waffe Mit welcher Waffe wurde das Lebewesen eliminiert?
   * (TWaffe/null)
   */
  public void ende(boolean tot, TWaffe waffe) {
```

```
    // Nur Todesröcheln, wenn er stirbt...
    if (tot) {
269   TPartikelVerwaltung partverwaltung =
       TSharedObjects.getPartikelVerwaltung();
      TAnzeige anzeige = TSharedObjects.getAnzeige();
272
      if (anzeige.fspieler != null) {
       anzeige.fspieler.addScore(ffeinddef.getPunkte());
275   }

      partverwaltung.startEffekt(waffe.getPartikelTot(), getMitte());
278   partverwaltung.startEffekt(TPartikelVerwaltung.Partikel.Score,
       getMitte(), "" + ffeinddef.getPunkte());
      if (waffe.getBeben() > 0) {
281    anzeige.setBeben(waffe.getBeben());
      }
      TSound.play(waffe.getTotSound());
284
      if (fitem != null) {
       fitem.setKoord(getMitte());
287    anzeige.addItem(fitem);
      }

290   if (ffeinddef.getDauersound() != null) {
       TSound.stoppe(ffeinddef.getDauersound());
      }
293  }
     super.ende(tot, waffe);
   }
296
}
```

Listing 10: TFeindDef.java

```
package de.rccc.java.witchcraft;

3 import javax.xml.parsers.*;
import org.w3c.dom.*;
import java.util.*;
6
/**
 * Die Definition eines Feindes, der "Bauplan" fuer einen Feind
9 */
public class TFeindDef {
 /**
12  * Liste der verfügbaren Feinddefinitionen (static)
  */
 protected static Map<String, TFeindDef> ffeinddefliste =
15  new HashMap<String, TFeindDef>();

 /**
18  * Welche Animation hat der Feind?
  */
 protected String fdarstellung = null;
21
 /**
  * Wieviel HP hat der Feind
24  */
 protected int flebenspunkte = 1;
```

```
27  /**
     * Welche Waffe hat der Feind?
     */
30  protected TWaffe fwaffe = null;

    /**
33   * Wiviele Punkte gibt der Feind
     */
    protected int fpunkte = 0;
36
    /**
39   * Welchem Bewegungsmuster folgt der Feind
     */
    protected int fmuster = 0;

42  /**
     * Wo wird die Waffe ausgeworfen?
     */
45  protected TVektor fwaffenausgang = null;

    /**
48   * Welcher Sound soll geloopt werden, wenn der Gegner kommt?
     * Bei Boss-Gegnern brauchbar, damit damit eine bedrohliche
     * Kulisse geschaffen werden kann.
51   */
    protected String fdauerSound = null;

54  /**
     * Konstruktor
     *
57   * @param darstellung Welche Animation für diesen Feind
     * @param lebenspunkte Wieviele lebenspunkte hat der Feind
     * @param waffe Welche Waffe hat der Feind
60   * @param punkte Weiviele Punkte gibt es für den Abschuss dieses Gegners
     * @param muster Welchem Muster folgt der Feind
     * @param waffenausgang Wo kommt die Waffe raus?
63   * @param dauersound Welcher Sound soll angefangen werden zu loopen, wenn
     * der Gegner aktiviert wird?
     */
66  TFeindDef(String darstellung, int lebenspunkte, String waffe,
     int punkte, int muster, TVektor waffenausgang, String dauersound) {

69   fdarstellung = darstellung;
     flebenspunkte = lebenspunkte;
     fwaffe = TWaffe.getWaffe(waffe);
72   fpunkte = punkte;
     fmuster = muster;
     fwaffenausgang = waffenausgang;
75   fdauerSound = dauersound;

     // Noch was korrigieren. Die Mitte des Bildes
78   // muss bei waffenausgang sein
     String waffenbild = fwaffe.getBild();
     if ((waffenbild != null) && (!waffenbild.equals(""))) {
81    TVektor groesse = TSharedObjects.getNewBild(waffenbild).groesse();
      groesse.mult(0.5);
      fwaffenausgang.sub(groesse);
84   }
```

```
};

/**
 * Gibt die Lebenspunkte dieses Feindes zurück
 */
public int getLebenspunkte() {
 return flebenspunkte;
}

/**
 * Gibt die Darstellung zurück
 */
public String getDarstellung() {
 return fdarstellung;
}

/**
 * Gibt die Waffe zurück
 */
public TWaffe getWaffe() {
 return fwaffe;
}

/**
 * Ginbt die Punkte des Feindes zurück
 */
public int getPunkte() {
 return fpunkte;
}

/**
 * Gibt das Bewegungsmuster des Feindes zurück
 */
public int getMuster() {
 return fmuster;
}

/**
 * Gibt die Startkoordinaten der Gesschosse zurück
 */
public TVektor getWaffenausgang() {
 return fwaffenausgang;
}

/**
 * Gibt den Sound zurueck, der mit dem Aktivieren des Gegners
 * gestartet werden soll
 */
public String getDauersound() {
 return fdauerSound;
}

/**
 * Fügt eine Feinddefinition in die List hinzu
 * @param id Unter welcher ID soll die Definition abgelegt werden
 * @param feinddef Diese Definition soll gespeichert werden
 */
public static void addFeindDef(String id, TFeindDef feinddef) {
 ffeinddefliste.put(id, feinddef);
```

```
144  }

     /**
147   * Holt eine Feinddefinition zu einer ID
      * @param id Die ID, unter der nachgeschaut werden soll
      */
150  public static TFeindDef getFeindDef(String id) {
      return ffeinddefliste.get(id);
     }
153
     /**
      * Leert die Liste (wenn neuer Level)
156   */
     public static void leereListe() {
      ffeinddefliste.clear();
159  }

     /**
162   * Erzeugt eine neue Feinddefiniton aus den Daten eines XML-Nodes
      * @param n Der Node, aus dem die Feinddefinition gebaut werden soll
      */
165  public static boolean addFeindDefAusNode(Node n) {
      String bild = null;
      String dauersound = null;
168   int lebenspunkte = -1;
      String waffenName = null;
      String id = null;
171   int punkte = -1;
      int muster = -1;
      TVektor waffenausgang = new TVektor();
174
      // in die Liste eintauchen
      NodeList nds = n.getChildNodes();
177   // wie aktuell ;)
      Node a;

180   // Einlesen der Daten
      for (int i = 0; i < nds.getLength(); i++) {
       a = nds.item(i);
183
       if (a.getNodeName().equals("bild")) {
        bild = a.getTextContent();
186    } else if (a.getNodeName().equals("ID")) {
        id = a.getTextContent();
       } else if (a.getNodeName().equals("leben")) {
189     lebenspunkte = Integer.parseInt(a.getTextContent());
       } else if (a.getNodeName().equals("waffe")) {
        waffenName = a.getTextContent();
192    } else if (a.getNodeName().equals("punkte")) {
        punkte = Integer.parseInt(a.getTextContent());
       } else if (a.getNodeName().equals("muster")) {
195     muster = Integer.parseInt(a.getTextContent());
       } else if (a.getNodeName().equals("waffenausgang")) {
        waffenausgang.leseAusXmlNode(a);
198    } else if (a.getNodeName().equals("dauersound")) {
        dauersound = a.getTextContent();
       }
201   }
```

```
    // Plausibilitaetspruefung
204 if ((bild == null) || (lebenspunkte <= 0) || (waffenName == null) ||
     (punkte < 0) || (muster == -1)){
     System.out.println("Fehlende Angaben (" + bild + ", " +
207   lebenspunkte + ", " + waffenName + ", " + punkte +
      ", " + muster + ")");
     return false;
210  }

    if (!TSharedObjects.bildInListe(bild)) {
213  System.out.println("Bild des Feindes \"" + id + "\": \"" +
      bild + "\" nicht gefunden.");
     return false;
216  }

    if (TWaffe.getWaffe(waffenName) == null) {
219  System.out.println("Waffe des Feindes \"" + id + "\": \"" +
      waffenName + "\" nicht gefunden.");
     return false;
222  }

    System.out.println(" Lade: " + id);
225 addFeindDef(id, new TFeindDef(bild, lebenspunkte, waffenName, punkte,
     muster, waffenausgang, dauersound));
    return true;
228 }
}
```

Listing 11: TGeschoss.java

```
package de.rccc.java.witchcraft;
2
import java.awt.geom.AffineTransform;
import java.awt.image.BufferedImage;
5
/**
 * Alle Geschosse (reduzieren Leben) erben hiervon, oder sind dieses.
8 * Geschosse sind die Repraesentation der Objekte, die rumfliegen.
 * Die Geschosse haben einen Link zu den definierten Waffen,
 * wo diese Geschosse genauer beschrieben sind.
11 */
public class TGeschoss extends TBildObjekt {
 /**
14  * Die Ausrichtung des Geschosses, in Grad
  */
 protected double fausrichtung;
17
 /**
  * Für welche Seite fliegt das Geschoss?
20  */
 protected int fseite;

23 /**
  * Time to life - wieviele ticks darf das
  * Geschoss überleben?
26  * Positiver fttl wird bei bewegen() runtergezählt
  * -1 - ewig
  * 0  - beim nächsten durchlauf löschen
29  */
```

```
protected int fttl = -1;

/**
 * Das ist ein Geschoss der Waffe..
 */
protected TWaffe fwaffe = null;

/**
 * Sicherheitszone für die ausserhalbBildschirm.
 * Notwendig, da der Kollsionsbereich nicht das gesammte
 * Objekt bedekt, sondern ein Quadrat in der Mitte.
 * Um zu verhindern, das das Geschoss dann am Rand einfach
 * verschwindet, wird das benuzt.
 */
private double fsicherheit = 0;

/**
 * Einmal das Bild holen reicht
 */
protected BufferedImage fimg = null;

/**
 * Offset für das Zeichnen
 */
protected TVektor foffset = null;

/**
 * Berechnet das notwendige zum Zeichnen des Bildes:
 * img, offset
 */
protected void berechneBildangaben() {
 if (fbild != null) {
  // Rail z.B. hat kein Bild
  fimg = fbild.getFrame(fausrichtung);

  foffset = new TVektor((fimg.getWidth() - fdim.x) / 2,
   (fimg.getHeight() - fdim.y) / 2);
 }
}

/**
 * Konstruktor des Geschosses
 *
 * @param koord Die Startkoordinaten
 * @param dim Die Groesse der Bounding-Box
 * @param geschw Der Geschwindigkeitsvektor
 * @param seite Die Gesinnung
 * @param waffe Das ist ein Geschoss der Waffe
 */
TGeschoss(TVektor koord, TVektor dim, TVektor geschw,
 int seite, TWaffe waffe) {

 super(koord, dim, geschw, waffe.getBild());

 // Der Kollsionsbereich ist ein kleines Rechteck in der Mitte
 // So hat man einen brauchbaren Kollisonsbereich, und es sieht
 // "cooler" aus.
 if (fbild != null) {
  // Rail hat kein Bild!
```

```
89      fdim.set(fbild.groesse(0));
        if (fdim.x > fdim.y) {
         fsicherheit = fdim.x;
92       fdim.x = fdim.y;
        } else {
         fsicherheit = fdim.y;
95       fdim.y = fdim.x;
        }
       }
98
       fwaffe = waffe;
       fwaffe.statIncAbgefeuert();
101
       // Ausrichtung ausrechnen
       fausrichtung = TVektor.genormt.winkel(geschw);
104
       berechneBildangaben();
       fseite = seite;
107    TSound.play(waffe.getStartSound());
      }

110   /**
       * Methode zum Zeichnen des Geschosses, ueberschreibt Methode
       * von TObjekt. Objekt wird automatisch in die Richtung gedreht,
113    * in die es fliegt
       *
       * @param g Das Grafik-Objekt, auf dem gezeichnet werden soll
116    */
      public void zeichne(java.awt.Graphics g) {
       // Debug: zeichnet den Kollisionsbereich des Geschosses
119    /*
       g.setColor(java.awt.Color.CYAN);
       g.fillRect((int)x,(int)y,(int)fimg.getWidth(),(int)fimg.getHeight());
122    g.setColor(java.awt.Color.RED);
       g.fillRect((int)fkoord.x,(int)fkoord.y,(int)fdim.x,(int)fdim.y);
       */
125
       g.drawImage(fimg, (int)(fkoord.x - foffset.x),
        (int)(fkoord.y - foffset.y), null);
128   }

      /**
131    * Ein Geschoss soll ein Lebewesen nur beruehren koennen, wenn
       * beide verfeindet sind
       *
134    * @param anderes Das Lebewesen, das das Geschoss potentiell beruehrt
       */
      public boolean beruehrt(TLebewesen anderes) {
137    return (anderes.getSeite() != fseite) && super.beruehrt(anderes);
      }

140   /**
       * Bewegt das Objekt einen Schritt, abhaengig von seiner Geschwindigkeit
       * Außerdem wird hier noch der TTL (Time to live) reduziert
143    */
      public void bewege() {
       super.bewege();
146    if (fttl > 0) {
        fttl -= 1;
```

```
    }
149 }

    /**
152  * Wird aufgerufen, wenn das Objekt gelöscht (aus den Listen) wird.
     * So kann jedes erbende Objekt ein letztes Röcheln abgeben.
     *
155  * @param tot True, wenn das Objekt wirklich stirbt.
     * False, wenn das Objekt nur "getötet" wird, weil es außerhalb
     * des Bildschirms ist
158  */
    public void ende(boolean tot) {
     if (tot) {
161   TSound.play(fwaffe.getHitSound());
      TSharedObjects.getPartikelVerwaltung().startEffekt(
       fwaffe.getPartikelTreff(),
164    getMitte(), getGeschw());
      fwaffe.statIncTreffer();
     }
167 }

    /**
170  * Ist das Geschoss am Ende? (ttl=0)
     *
     * @return true - Geschoss aus der Liste nehmen
173  */
    public boolean tot() {
     return fttl == 0;
176 }

    /**
179  * Gibt die Trefferpunkte des Geschosses zurück
     */
    public int getTrefferp() {
182  return fwaffe.getSchaden();
    }

185 /**
     * Gibt die TTL (time to life) zurück.
     * Wenn =0, dann aus der Liste nehmen
188  */
    public int getTtl() {
     return fttl;
191 }

    /**
194  * Gibt die Waffe zurück, aus dem das Geschoss gekommen ist
     */
    public TWaffe getWaffe() {
197  return fwaffe;
    }

200 /**
     * Überprüft, ob das Objekt nicht jetzt ausgespielt hat.
     * Hierhin verlagert, weil das Geschoss nur einen kleinen
203  * Kollisionsbereich in der Mitte hat,
     * und es so einfach "verschwinden" würde.
     * Daher hier etwas toleranz aufbauen... :)
206  *
```

```
   * @return true, wenn das Ojekt nicht mehr gebraucht wird
   */
209  public boolean ausserhalbBildschirm () {
   int x = TSharedObjects.FENSTER_BREITE;
   int y = TSharedObjects.FENSTER_HOEHE;
212  TVektor ru = getRU();
   return (ru.x < -fsicherheit) || (ru.y < -fsicherheit) ||
    (fkoord.x > x+fsicherheit) || (fkoord.y > y+fsicherheit);
215  }

}
```

Listing 12: TItem.java

```
package de.rccc.java.witchcraft;
2
import javax.xml.parsers.*;
import org.w3c.dom.*;
5
/**
 * Klasse für alle vom Spieler aufsammelbaren Items.
8 * Diese beschraenken sich auf ein "Health"-Objekt
 * und die verschiedenen Waffen, waere aber erweiterbar.
 */
11 public class TItem extends TBildObjekt {
 /**
  * Die Items, die zur Verfuegung stehen.
14  * Heilung:  Heilt den Spieler
  */
 static public enum Items {
17   /**
   * Not in List. Fehlwert
   */
20  NIL,
   /**
    * Heilung.
23  * Dieses Item heilt den Spieler
    */
   Heilung,
26  /**
    * Waffe.
    * Item ist eine Waffe
29  */
   Waffe
 };
32
 /**
  * Das Item, das das Objekt ist
35  */
 protected Items fitem;

38 /**
  * Der int-Parameter zu dem Item.
  * Z.B. heilt ein Heilung fiparam Punkte
41  */
 protected int fiparam = 0;

44 /**
  * Der String-Param zu diesem Item
```

```
     */
47   protected String fsparam = null;

     /**
50    * Konstuktor.
      *
      * @param koord Der Koordinatenvektor
53    * @param dim Der Groessenvektor
      * @param geschw Der Geschwindigkeitsvektor
      * @param darstellung Darstellung des Objekts
56    * @param item Das Item, das das Objekt ist
      * @param iparam Integer-Parameter des Items
      * @param sparam String-Parameter des Items
59    */
     TItem(TVektor koord, TVektor dim, TVektor geschw, String darstellung,
      Items item, int iparam, String sparam) {
62
      super(koord, dim, geschw, darstellung);
      fitem = item;
65    fiparam = iparam;
      fsparam = sparam;
     }
68
     /**
      * Von welchem Typ ist dieses Item?
71    *
      * @return Das Item
      */
74   public Items getItem() {
      return fitem;
     }
77
     /**
      * Der Integer-Parameter des Item.
80    * z.B. Die Healthanzahl.
      *
      * @return Der Parameter
83    */
     public int getiParam() {
      return fiparam;
86   }

     /**
89    * Der String-Parameter des Item.
      * z.B. Die Waffe.
      *
92    * @return Der Parameter
      */
     public String getsParam() {
95    return fsparam;
     }

98   /**
      * Generiert ein Item aus dem übergebenen XML-Node
      *
101   * @param nds Die Nodeliste des Items
      */
     public static TItem itemAusNode(NodeList nds) throws Exception {
104   String bild = null;
```

```
    TItem.Items item = TItem.Items.NIL;
    int iparam = 0;
107 String sparam = null;

    for (int j = 0; j < nds.getLength(); j++) {
110  Node a = nds.item(j);
     if (a.getNodeName().equals("bild")) {
      bild = a.getTextContent();
113  } else if (a.getNodeName().equals("item")) {
      item = TItem.Items.valueOf(a.getTextContent());
     } else if (a.getNodeName().equals("iparam")) {
116   iparam = Integer.parseInt(a.getTextContent());
     } else if (a.getNodeName().equals("sparam")) {
      sparam = a.getTextContent();
119  }
    }

122 if ((bild==null) || (item == TItem.Items.NIL)) {
     throw new Exception("Einem Item fehlt was");
    }
125
    if (!TSharedObjects.bildInListe(bild)) {
     throw new Exception("Das ItemBild \"" + bild + "\" gibt es nicht.");
128 }

    return new TItem(null, null, new TVektor(-4, 0), bild, item,
131  iparam, sparam);
  }

134 /**
   * Überprüft, ob das Objekt nicht jetzt ausgespielt hat.
   * Überschreiben, weil Items von rechts nach links durchlaufen,
137 * und wenn zufällig ein Feind außerhalb des Bildschirms eleminiert
   * wird, dann würde das Item verschwinden.
   *
140 * @return true, wenn das Ojekt nicht mehr gebraucht wird
   */
  public boolean ausserhalbBildschirm () {
143 int x = TSharedObjects.FENSTER_BREITE;
   int y = TSharedObjects.FENSTER_HOEHE;
   TVektor ru = getRU();
146 return (ru.x < 0) || (ru.y < 0);
  }
}
```

Listing 13: TLebewesen.java

```
package de.rccc.java.witchcraft;
2

import java.util.*;

5 /**
  * Lebewesen sind alle Einheiten, die Leben haben
  */
8 public class TLebewesen extends TBildObjekt {
  /**
   * Anzahl der Lebenspunkt des Lebewesens
11 */
  protected double flebenspunkte;
```

```
14  /**
     * Maximale Lebendspunkte des Lebewesens.
     * wichtig für die Berechnung des Balkens
17   */
    protected double flebenmax;

20  /**
     * Die Liste aller Objekte, die bearbeitet werden müssen,
     * wenn das hier zerstört wird
23   */
    protected List<TObjekt> fabhaengige = new LinkedList<TObjekt>();

26  /**
     * Auf welcher Seite ist dieses Lebewesen? 0 = Spieler
     */
29  protected int fseite;

    /**
32   * Balkenlaenge.
     * Damit das nicht jedesmal bei der Anzeige berechnet werden muss
     */
35  protected int fbll;

    /**
38   * ROF (rate of fire) - die Schussfrequenz
     */
    protected int frof;
41
    /**
     * Konstruktor
44   *
     * @param koord Die Koordinaten, auf dem das Lebewesen erscheint.
     * @param dim Die Dimensionen des Lebewesen. bei null automatische
47   * Erkennung
     * @param geschw die Bewegung des Lebewesens
     * @param darstellung Welches Bild nutzt das Lebewesen
50   * @param lebenspunkte Die Lebendspunkte des Lebewesens
     * @param seite welcher Seite gehört das Lebewesen an
     */
53  TLebewesen(TVektor koord, TVektor dim, TVektor geschw, String darstellung,
     double lebenspunkte, int seite) {

56   super(koord, dim, geschw, darstellung);
     flebenspunkte = lebenspunkte;
     flebenmax = lebenspunkte;
59   fseite = seite;
     // fbll berechnen
     treffer(0);
62   frof = 0;
    }

65  /**
     * Zieht die Trefferpunkte von dem Leben ab, und gibt zurück, ob
     * der gestorben ist
68   *
     * @param punkte Die Punkte, die abgezogen werden
     * @return True, wenn der gestorben ist
71   */
```

```
 public boolean treffer(double punkte) {
  flebenspunkte -= punkte;
  // Falls der Spieler Heilung aufsammelt
  if (flebenspunkte > flebenmax) {
   flebenspunkte = flebenmax;
  }
  fbll = (int) ((fdim.x-2)*(flebenspunkte/flebenmax));
  return flebenspunkte < 0;
 }

 /**
  * Zeichnet das Bild an seinen Koordinaten auf der Zeichenflaeche g.
  * Zusätzlich wird noch ein Health Balken gemalt.
  *
  * @param g Die Zeichenflaeche, auf der das Objekt gezeichnet werden soll
  */
 public void zeichne(java.awt.Graphics g) {
  super.zeichne(g);
  g.setColor(java.awt.Color.BLACK);
  g.fillRect((int)fkoord.x, (int)fkoord.y-3, (int)fdim.x, 3);
  g.setColor(java.awt.Color.RED);
  g.fillRect((int)fkoord.x+1, (int)fkoord.y-2, fbll, 2);
 }

 /**
  * Fügt ein Objekt an die Liste der abhängigen Objekte hinzu
  */
 public void addAbhaengige(TObjekt abhaengig){
  fabhaengige.add(abhaengig);
 }

 /**
  * Löscht ein Objekt aus der Liste der abhängigen Objekte.
  * Wenn z.B. eine Rakete eingeschlagen hat,
  * ist es nicht mehr abhängig :)
  */
 public void delAbhaengige(TObjekt abhaengig){
  fabhaengige.remove(abhaengig);
 }

 /**
  * Gibt eine Liste der abhängigen Objetke zurück.
  * Z.B. Werden die Raketen, die dieses Objekt als Ziel haben,
  * hier gespeichert. Wenn dieses Objekt stirbt, werden die
  * Raketen auf Dumb-Fire gestellt
  */
 public List<TObjekt> getAbhaengige() {
  return fabhaengige;
 }

 /**
  * Gibt die Seite zurück, der dieses Lebewesen angehört
  */
 public int getSeite() {
  return fseite;
 }

 /**
  * Wird aufgerufen, wenn das Objekt gelöscht (aus den Listen) wird.
```

```
131     * So kann jedes erbende Objekt ein letztes Röcheln abgeben.. :)
        *
        * @param tot True, wenn das Objekt wirklich stirbt.
134     * False, wenn das Objekt nur "getötet" wird, weil es außerhalb
        * des Bildschirms ist
        * @param waffe Mit welcher Waffe wurde das Lebewesen
137     * eliminiert? (TWaffe/null)
        */
      public void ende(boolean tot, TWaffe waffe) {
140     // Immer alle Abhängigen Objekte überarbeiten
       for (TObjekt o: fabhaengige) {
        if (o instanceof TRakete) {
143       ((TRakete)o).setZiel(null);
        }
       }
146    fabhaengige.clear();
      }

149   /**
       * Liefert die Lebenspunkte des Lebewesens
       */
152   public double getLeben() {
       return flebenspunkte;
      }
155
      /**
       * Liefert die maximale Anzahl der Lebenspunkte des Lebewesens
158    */
      public double getLebenMax() {
       return flebenmax;
161   }

      /**
164    * Setzt die Lebenspunkte des Lebewesens
       */
      public void setLeben(double leben) {
167    flebenspunkte = leben;
       treffer(0);
      }
170
      /**
       * Setzt die maximalen Lebenspunkte des Lebewesens
173    */
      public void setLebenMax(double lebenmax) {
       flebenmax = lebenmax;
176   }
    }
```

Listing 14: TLevel.java

```
    package de.rccc.java.witchcraft;

3   import java.util.*;
    import javax.xml.parsers.*;
    import org.w3c.dom.*;
6
    /**
     * Klasse, die fuer das Laden, repraesentieren (Modell) und Darstellen
9    * des Levels sowie eventueller grafischen Meldungen zustaendig ist.
```

```
 */
public class TLevel {
 /**
  * Klasse fuer ein einzelnes Levelsegment: der kleinste unteilbare
  * Teil eines Levels
  */
 private class TLevelSegment {
  /**
   * Bild fuer das Levelsegment
   */
  private TAnimation fdarstellung;

  /**
   * Meldung fuer den Spieler, die beim Aktivieren des Segments
   * einmalig angezeigt werden soll
   */
  private String fmeldung = null;

  /**
   * Wurden die Feinde aus der lokalen Liste schon aktiviert?
   */
  private boolean faktiv;

  /**
   * Ambiente-Sounds
   */
  private String fsound = null;

  /**
   * Liste der Feinde, die mit diesem
   * Segment erscheinen sollen. Diese "schlafen" hier bis
   * das Segment sichtbar wird.
   */
  private List<TFeind> ffeinde = new LinkedList<TFeind>();

  /**
   * Konstruktor
   * @param darstellung Das Bild fuer das Levelsegment
   */
  TLevelSegment(TAnimation darstellung) {
   fdarstellung = darstellung;
   faktiv = false;
  }

  /**
   * Lebewesen zur Liste zufuegen
   */
  public void addFeind(TFeind f) {
   ffeinde.add(f);
  }

  /**
   * Liefert die Liste der Lebewesen
   */
  public List<TFeind> getFeinde() {
   return ffeinde;
  }

  /**
```

```
69     * Zeichnet das Levelsegment
       */
      public void zeichne(java.awt.Graphics g, int offset) {
72     g.drawImage(fdarstellung.getFrame(), offset, 0, null);
      }

75    /**
       * Fuege die lokalen, schlafenden Feinde in die
       * eigentliche Feinde-Liste ein und aktiviere eine eventuell
78     * in diesem Segment vorhandene Meldung
       */
      public void aktiviere() {
81     TAnzeige anzeige = TSharedObjects.getAnzeige();
       if (!faktiv) {
        faktiv = true;
84      anzeige.getLebewesen().addAll(ffeinde);

        // irgendwelche geloopten Sounds starten
87      for (TFeind feind: ffeinde) {
         if (feind.getFeindDef().getDauersound() != null) {
          TSound.playMusik(feind.getFeindDef().getDauersound());
90       }
        }

93      // Meldung anzeigen und Spiel pausieren, falls vorhanden
        if (fmeldung != null) {
         setAktuelleMeldung(fmeldung);
96       anzeige.pause();
        }

99      // Ambientesound spielen
        TSound.play(fsound);
       }
102   }

      /**
105    * Setze die Meldung, die beim Aktivieren des Segments
       * einmal angezeigt werden soll
       */
108   public void setMeldung(String meldung) {
       fmeldung = meldung;
      }
111
      /**
       * Setzen den Sound, der abgespielt werden soll.
114    * Als "Ambiente"
       */
      public void setSound(String sound) {
117    fsound = sound;
      }
     }
120
     /**
      * Name des Levels
123   */
     private String fname;

126  /**
      * Liste aller Segmente, die das Level ausmachen
```

```
     */
129  private LinkedList<TLevelSegment> flevel =
      new LinkedList<TLevelSegment>();

132  /**
      * Da immer nur eine Meldung aktiv sein kann, kann sich das
      * Level direkt darum kuemmern
135   */
     private TBildObjekt faktuelleMeldung;

138  /**
      * Breite in Pixel eines Segments. Da diese fuer alle Segmente
      * gleich ist, braucht die Unterklasse TLevelSegment davon nichts
141   * zu wissen
      */
     private static int fsegmentBreite = 80;
144
     /**
      * Die Anzahl der Segmente, die nebeneinander auf der Spielflaeche
147   * Platz haben
      */
     private static int fanzahlSegmente;
150
     /**
      * Wie weit verschoben ist das linkeste sichtbare Levelsegment
153   */
     private int foffset = 0;

156  /**
      * Nummer des Levels, das geladen wurde
      */
159  private int flevelNummer;

     /**
162   * Zaehlt die Anzahl der Segmente, an denen der Spieler schon
      * vorbeigeflogen ist. Diese Zahl wird in das Savegame gespeichert
      */
165  private int fgeloeschteSegmente = 0;

     /**
168   * Anzahl der "aufgefuellten" Segmente - diese sollen beim
      * erneuten Speichern nicht nochmal mitgespeichert werden
      */
171  private int fpadSegmente = 0;

     /**
174   * Map fuer die Segmentgrafiken (gleiche Segmente koennen das
      * selbe TAnimation-Objekt zur Darstellung verwenden)
      */
177  private Map<String, TAnimation> segmentGrafiken =
      new HashMap<String, TAnimation>();

180  /**
      * Konstruktor mit Level-Nummer
      * @param nummer Die Nummer des Levels, das geladen werden soll
183   */
     public TLevel(int nummer) throws Exception {
      this(nummer, 0, false);
186  }
```

```
    /**
189  * Konstruktor mit Level-Nummer und Anzahl der Segmente, die
     * nicht geladen werden sollen
     * @param nummer Die Nummer des Levels, das geladen werden soll
192  * @param ueberspring Ignoriere diese Anzahl von Segmenten im Level
     * @param pad Wurde das Level geladen und soll vorne aufgefuellt
     * werden: (Gegner sollen an vorgesehenen Positionen
195  * starten, nicht alle auf einmal, ausserdem steckt der Spieler
     * nicht direkt im Gemetzel)
     */
198  public TLevel(int nummer, int ueberspring, boolean pad)
     throws Exception {

201  faktuelleMeldung = null;
     fanzahlSegmente = TSharedObjects.FENSTER_BREITE / fsegmentBreite;
     flevelNummer = nummer;
204
     String levelDatei = "/level/level" + nummer + ".xml";
     java.net.URL pfad = TLevel.class.getResource(levelDatei);
207
     System.out.println("Lade Level " + nummer + ": "+levelDatei);

210  if (pfad == null) {
      throw new Exception(levelDatei + " nicht gefunden");
     }
213
     //vorbereiten: Jeder Level hat seine eigenen Feinde
     TFeindDef.leereListe();
216
     String nodeName;
     Document config = DocumentBuilderFactory
219       .newInstance().newDocumentBuilder()
          .parse(pfad.toURI().toString());
     NodeList nds = config.getChildNodes();
222
     if (!nds.item(0).getNodeName().equals("level")) {
      throw new Exception("Fehlerhafte Leveldatei: " + levelDatei +
225     " (<level> als Root-Element erwartet)");
     }

228  // Level wurde geladen, wir wollen erstmal auffuellen
     if (pad) {
      fpadSegmente = fanzahlSegmente;
231   segmentGrafiken.put("1", new TAnimation("level" +
       nummer + "seg1.gif"));
      for (int i = 1; i <= fanzahlSegmente; i++) {
234    flevel.add(new TLevelSegment(segmentGrafiken.get("1")));
      }
     }
237
     // jetzt Nodes verarbeiten
     nds = nds.item(0).getChildNodes();
240  for (int i = 0; i < nds.getLength(); i++) {
      if (nds.item(i) instanceof Element) {
       nodeName = nds.item(i).getNodeName();
243    if (nodeName.equals("name")) {
        fname = nds.item(i).getTextContent();
       } else if (nodeName.equals("struktur")) {
```

```
246         ladeSegmente(nds.item(i).getChildNodes(), nummer, ueberspring);
            } else if (nodeName.equals("spieler")) {
             ladeSpielerDaten(nds.item(i).getChildNodes());
249         } else if (nodeName.equals("feinde")) {
             ladeFeindDefs(nds.item(i).getChildNodes());
            }
252        }
       }

255    if (!pad) {
        // die ersten Segemente auf dem Bildschirm aktivieren
        for (int i = 0; i < fanzahlSegmente; i++) {
258      flevel.get(i).aktiviere();
        }
       }
261  }

     /**
264   * Setzt die aktuell anzuzeigende Meldung an einer zentrierten Position
      * @param meldung Der Eintrag der Bilderliste, der angezeigt werden soll
      */
267  public void setAktuelleMeldung(String meldung) {
      // Wenn Spieler Tot, dann keine weitere (Level-)Meldung
      if (TSharedObjects.getAnzeige().fspieler == null) {
270    return; // Nur mt Protest!
      }
      TAnimation bild = TSharedObjects.getBild(meldung);
273   int breite = (int)bild.groesse().x;
      int hoehe = (int)bild.groesse().y;
      faktuelleMeldung = new TBildObjekt(
276    new TVektor((TSharedObjects.FENSTER_BREITE / 2) - (breite / 2),
          (TSharedObjects.FENSTER_HOEHE / 2) - (hoehe / 2)),
       null, null, meldung);
279  }

     /**
282   * Laedt ein Level aus einer Nodeliste (struktur)
      * @param nds Die Nodeliste der Segmente
      * @param nummer Die Nummer des Levels
285   * @param ueberspring Die Anzahl der Segmente, die ignoriert werden sollen
      */
     private void ladeSegmente(NodeList nds, int nummer,
288   int ueberspring) throws Exception {

      String nodeName;
291
      for (int i = 0; i < nds.getLength(); i++) {
       String segmentInhalt  = null;
294    String segmentMeldung = null;
       String segmentSound   = null;

297    if (nds.item(i) instanceof Element) {
        nodeName = nds.item(i).getNodeName();
        // Wir interessieren uns nur fuer Segmente, und nur dann,
300     // wen wir schon genug uebersprugngen haben (beim Laden)
        if ((nodeName.equals("segment")) && (--ueberspring < 0)) {
         for (int j = 0; j < nds.item(i).getChildNodes().getLength(); j++) {
303       if (nds.item(i).getChildNodes().item(j)
           .getNodeName().equals("nummer")) {
```

```
        segmentInhalt = nds.item(i).getChildNodes()
306       .item(j).getTextContent();
       } else if (nds.item(i).getChildNodes().item(j)
         .getNodeName().equals("meldung")) {
309      segmentMeldung = nds.item(i).getChildNodes()
         .item(j).getTextContent();
       } else if (nds.item(i).getChildNodes().item(j)
312        .getNodeName().equals("sound")) {
        segmentSound = nds.item(i).getChildNodes()
         .item(j).getTextContent();
315    }
      }

318   // Wenn wir das Segment noch nicht kennen, erzeugen wir es...
      if (segmentGrafiken.get(segmentInhalt) == null) {
       segmentGrafiken.put(segmentInhalt,
321     new TAnimation("level" + nummer + "seg" +
        segmentInhalt + ".gif"));
      }
324
      TLevelSegment tmpseg = new TLevelSegment(
       segmentGrafiken.get(segmentInhalt));
327   tmpseg.setMeldung(segmentMeldung);
      tmpseg.setSound(segmentSound);

330   // ...ermitteln seine Feinde...
      verarbeiteSegment(nds.item(i).getChildNodes(), nummer, tmpseg);
      // ...und fuegen es dem Level zu
333   flevel.add(tmpseg);
     } else if (!nodeName.equals("segment")) {
      throw new Exception("Fehler beim Laden von Level " +
336     "level/level" + nummer + ".xml (<segment> erwartet)");
     }
    }
339 }
}

342 /**
 * Lädt Spielerangaben
 *
345 * @param nds Die Nodeliste des Spielers
 */
private void ladeSpielerDaten(NodeList nds) throws Exception {
348 for (int j = 0; j < nds.getLength(); j++) {
  Node a = nds.item(j);
  if (a.getNodeName().equals("waffe")) {
351  TSharedObjects.getAnzeige().fspieler.addWaffe(a.getTextContent());
  }
 }
354

}

357 /**
 * Lädt die Feinddefinitionen
 *
360 * @param nds Die Nodeliste des Spielers
 */
private void ladeFeindDefs(NodeList nds) throws Exception {
363 System.out.println("Lade FeindDefinition");
```

```
   for (int j = 0; j < nds.getLength(); j++) {
    Node a = nds.item(j);
366
    if (a.getNodeName().equals("feind")) {
     if (!TFeindDef.addFeindDefAusNode(a)) {
369   throw new Exception("Fehler beim Laden der Feinddefinitionen");
     }
    }
372
   }
  }
375

  /**
378 * verarbeiteSegment fuegt die in der NodeList nds gefundenen Feinde
   * an die Liste des TLevelSegment seg an
   * @param nds Die zu verarbeitende NodeListe
381 * @param nummer Die Nummer des Levels (fuer Fehlerausgaben)
   * @param seg Das Segment, bei dem die Gegner zugefuegt werden sollen
   */
384 private void verarbeiteSegment(NodeList nds, int nummer,
   TLevelSegment seg) throws Exception {

387 String levelDatei = "level/level" + nummer + ".xml";
   String nodeName;

390 // alle Feinde durchgehen
   for (int i = 0; i < nds.getLength(); i++) {
    if ((nds.item(i) instanceof Element) &&
393   (nds.item(i).getNodeName().equals("feind"))) {

     String id = null;
396  int startpunkt = -1;
     TItem item = null;

399  for (int j = 0; j < nds.item(i).getChildNodes().getLength(); j++) {
      Node a = nds.item(i).getChildNodes().item(j);
      if (a.getNodeName().equals("startpunkt")) {
402    startpunkt = Integer.parseInt(a.getTextContent());
      } else if (a.getNodeName().equals("ID")) {
       id = a.getTextContent();
405   } else if (a.getNodeName().equals("item")) {
       if (item != null) {
        throw new Exception("Fehler in Leveldatei " + levelDatei +
408       " (Ein Feind kann nur ein Item haben)");
       }
       item = TItem.itemAusNode(a.getChildNodes());
411   }
     }

414  // Alle Angaben fuer den Feind muessen gemacht sein
     if ((id == null) || (startpunkt == -1)) {
      throw new Exception("Fehler in Leveldatei " + levelDatei +
417    " (id oder  startpunkt " +
       "fehlt bei feind in segment)");
     }
420
     // Ausserdem muss das Bild bereits geladen sein
     TFeindDef feinddef = TFeindDef.getFeindDef(id);
```

```
423       if (feinddef != null) {
           seg.addFeind(new TFeind(feinddef, startpunkt, 1, item));
          } else {
426        throw new Exception("Fehler in Leveldatei " + levelDatei +
            " (Feinddef '" + id + "' ist nicht definiert)");
          }
429      }
        }
       }

432
       /**
        * Level einen Tick weiter scrollen
435     */
       public void weiterScrollen() {
        foffset -= 4;
438     if (foffset < -fsegmentBreite) {
         foffset = 0;
         if (flevel.size() > (fanzahlSegmente + 1)) {
441       flevel.get(fanzahlSegmente + 1).aktiviere();
          flevel.removeFirst();
          fgeloeschteSegmente++;
444       if (fpadSegmente > 0) {
           fpadSegmente--;
          }
447      } else {
          // Wenn Level zuende ist, dann das vorderste hinten einketten.
          TLevelSegment vorne = flevel.get(0);
450       flevel.removeFirst();
          flevel.add(vorne);
         }
453     }
       }

456    /**
        * Ist das Level schon am Ende?
        *
459     * @return true, wenn das Level am Ende ist, und nur noch wiederholt wird
        */
       public boolean levelZuEnde() {
462     return flevel.size() <= (fanzahlSegmente + 1);
       }

465    /**
        * Level zeichnen
        */
468    public void zeichne(java.awt.Graphics g) {
        int count = 0;
        for (TLevelSegment ls: flevel) {
471      ls.zeichne(g, (count * fsegmentBreite) + foffset);

         count++;
474      if (count >= (fanzahlSegmente + 2))
          return;
        }
477    }

       /**
480     * Zeichnet die eventuell aktive Segmentmeldung
        */
```

```
  public void zeichneMeldung(java.awt.Graphics g) {
483   if (faktuelleMeldung != null) {
       faktuelleMeldung.zeichne(g);
      }
486   }

      /**
489    * Deaktiviere alle eventuell aktiven Meldungen
       */
      public void deaktiviereMeldung() {
492    faktuelleMeldung = null;
      }

495   /**
       * Liefert die Nummer des aktuell geladenen Levels
       */
498   public int getLevelNummer() {
       return flevelNummer;
      }
501
      /**
       * Liefert die Anzahl der Segmente, an denen der Spieler schon
504    * vorbeigeflogen ist - ausgenommen diese, die extra nach dem
       * Laden eines Savegames aufgefuellt wurden
       */
507   public int getGeloeschteSegmente() {
       int segs = fgeloeschteSegmente - fpadSegmente +
        fanzahlSegmente;
510    return (segs < 0) ? 0 : segs;
      }
    }
```

Listing 15: TMenuHTMLzeig.java

```
    package de.rccc.java.witchcraft;

3   import java.awt.*;
    import javax.swing.*;
    import java.awt.event.*;
6
    /**
     * Klasse, um einen neuen Frame mit einer HTML-Datei als Inhalt
9    * anzuzeigen
     */
    public class TMenuHTMLzeig {
12   /**
      * Konstruktor, der die gesamte Funktionalitaet enthaelt
      */
15   public TMenuHTMLzeig(String htmldatei) {
      final JFrame p = new JFrame("Witchcraft - Info");

18    try {
       java.net.URL url = TSound.class.getResource("/html/" + htmldatei);
       JEditorPane editorPane = new JEditorPane(url);
21     editorPane.setEditable(false);

       JScrollPane hauptscroller = new JScrollPane(editorPane);
24     hauptscroller.setMinimumSize(new Dimension(66, 400));
       hauptscroller.setSize(new Dimension(665, 400));
```

```
    hauptscroller.setPreferredSize(new Dimension(665, 400));
27
    p.add(hauptscroller, BorderLayout.CENTER);
    JButton weg = new JButton("Schließen");
30  weg.addActionListener(new ActionListener() {
     public void actionPerformed(ActionEvent e) {
      p.dispose();
33    TSharedObjects.getAnzeige().start();
     }
    });
36  p.add(weg, BorderLayout.SOUTH);
    //p.setSize(650,400);
   } catch (Exception e) {
39  p.add(new JLabel("Fehler: Konnte html nicht laden: " + e) );
   }

42 p.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
     TSharedObjects.getAnzeige().start();
45  }
   });

48 p.pack();
   p.setVisible(true);
  }
51}
```

Listing 16: TObjekt.java

```
package de.rccc.java.witchcraft;

3 import java.util.Map;
import java.awt.image.*;

6 /**
 * TObjekt stellt ein auf der Zeichenflaeche darstellbares Objekt dar.
 * Es hat Koordinaten-, Groessen-, und Geschwindigkeitsvektoren. Eine
9 * Darstellung hat TObjekt nicht, darum muessen sich erbende Klassen
 * kuemmern.
 */
12 abstract public class TObjekt {
 /**
  * Der Koordinatenvektor
15  */
 protected TVektor fkoord;

18 /**
  * Der Groessenvektor
  */
21 protected TVektor fdim;

 /**
24  * Der Vektor, der "Rechts Unten" anzeigt.
  * So müssen wir nicht mit Objekten um uns scheissen.
  * (neue TVektor-Objekte, die bei jedem mal Anfragen
27  * von "rechts unten" erzeugt werden muessten)
  */
 protected TVektor fru;
30
```

```
  /**
   * Der Geschwindigkeitsvektor
33 */
  protected TVektor fgeschw;

36 /**
   * Konstruktor
   *
39 * @param koord Der Koordinatenvektor
   * @param dim Der Groessenvektor
   * @param geschw Der Geschwindigkeitsvektor
42 */
  TObjekt(TVektor koord, TVektor dim, TVektor geschw) {
   this.fkoord  = koord;
45 this.fgeschw = geschw;
   this.fdim = dim;
   if ((fkoord != null) && (fdim != null)) {
48  this.fru = fkoord.newAdd(fdim);
   }
  }
51

  /**
   * Veraendere die Y-Komponente des Geschwindigkeitsvektors
54 * @param g Der Wert, der auf die Y-Komponente addiert werden soll
   */
  public void gravitationAdd(double g) {
57 this.fgeschw.y += g;
  }

60 /**
   * Multipliziere die Y-Komponente des Geschwindigkeitsvektors
   * mit einem Faktor
63 * @param g Der Wert, mit dem die Y-Komponente multipliziert werden soll
   */
  public void gravitationMult(double g) {
66 this.fgeschw.mult(g);
  }

69 /**
   * Bewegt das Objekt einen Schritt, abhaengig von seiner Geschwindigkeit
   */
72 public void bewege() {
   fkoord.add(fgeschw);
   fru.add(fgeschw);
75 }

  /**
78 * Setzt die Koordinaten des Objektes
   */
  public void setKoord(TVektor koord) {
81 this.fkoord = koord;
   // RU nachziehen
   this.fru = koord.newAdd(fdim);
84 }

  /**
87 * Liefert den Koordinatenvektor des Objekts
   */
  public TVektor getKoord() {
```

```
90    return fkoord;
    }

93    /**
     * Liefert den Geschwindigkeitsvektor des Objekts
     */
96    public TVektor getGeschw() {
     return fgeschw;
    }
99
    /**
     * Liefert die Mitte des Objektes
102   */
    public TVektor getMitte() {
     return new TVektor(fkoord.x + fdim.x / 2, fkoord.y + fdim.y / 2);
105 }

    /**
108  * Liefert den Vektor auf die rechte untere Ecke der
     * Boundingbox des Objekts
     */
111   public TVektor getRU() {
     // Eigentlich ist es Objektorientierter es so zu machen:
     // return fkoord.newAdd(fdim);
114  // Aber so scheissen wir zu viele Objekte
     return fru;
    }
117
    /**
     * Sind die Koordinaten innerhalb der von mir gebrauchten Fläche?
120  *
     * @param k Der Koordinatenvektor, der ueberprueft werden soll
     * @return true: Die Koordianten liegen innerhalb
123  */
    public boolean innerhalb(TVektor k) {
     return (fkoord.x <= k.x) && (fkoord.y <= k.y) &&
126    (fru.x >= k.x) && (fru.y >= k.y);
    }

129 /**
     * Zeichnet das Bild an seinen Koordinaten auf der Zeichenflaeche g
     *
132  * @param g Die Zeichenflaeche, auf der das Objekt gezeichnet werden soll
     */
    abstract public void zeichne(java.awt.Graphics g);
135
    /**
     * Ueberprueft, ob das Objekt mit einem Anderen kollidiert
138  *
     * @param anderes Das andere Objekt, mit dem auf Kollision geprueft
     * werden soll
141  * @return Besteht eine Kollision (ueberschneiden sich die Objekte)?
     */
    public boolean beruehrt(TObjekt anderes) {
144  TVektor aru = anderes.getRU();
     return !((anderes.fkoord.x > fru.x) ||
       (anderes.fkoord.y > fru.y) ||
147    (fkoord.x > aru.x) ||
       (fkoord.y > aru.y));
```

```
  }

  /**
   * Überprüft, ob das Objekt nicht jetzt ausgespielt hat
   *
   * @return true, wenn das Ojekt nicht mehr gebraucht wird
   */
  public boolean ausserhalbBildschirm() {
    int x = TSharedObjects.FENSTER_BREITE;
    int y = TSharedObjects.FENSTER_HOEHE;
    return (fru.x < 0) || (fru.y < 0) || (fkoord.x > x) || (fkoord.y > y);
  }

  /**
   * Wird aufgerufen, wenn das Objekt gelöscht (aus den Listen) wird.
   * So kann jedes erbende Objekt ein letztes Röcheln abgeben.
   * Methode ist nicht abstrakt, weil nicht jedes Objekt dessen Klasse von
   * TObjekt erbt, diese Methode benoetigt.
   *
   * @param tot True, wenn das Objekt wirklich stirbt.
   * False, wenn das Objekt nur "getötet" wird, weil es außerhalb
   * des Bildschirms ist
   */
  public void ende(boolean tot) {}
}
```

Listing 17: TOptionenFenster.java

```
package de.rccc.java.witchcraft;

import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import java.awt.event.*;

/**
 * Klasse, die ein Fenster darstellt, in dem der Spieler seinen Namen
 * eingeben und die Schwierigkeitsstufe auswahlen kann. Der Startknopf
 * ruft dann neuesSpiel() in TAnzeige auf.
 */
public class TOptionenFenster {
  /**
   * Der Schwierigkeitsgrad, der ausgewaehlt werden soll,
   * wird hier gespeichert
   */
  protected TSpieler.Schwierigkeitsgrade fschwierig =
    TSpieler.Schwierigkeitsgrade.normal;

  /**
   * Der Name des Spielers soll hier gespeichert werden
   */
  protected String fname = null;

  /**
   * Das Textfeld, in dem der Spieler seinen Namen eingeben kann.
   * Kann nicht lokal im Konstruktor liegen, da unsere feine Klasse
   * DL (d.h. der DocumentListener fuer das Feld) Zugriff braucht
   */
```

```
  protected JTextField fnameFeld = null;
33
  /**
   * DocumentListener ist dummerweise ein Interface,
36 * deswegen brauchen wir erst mal eine Klasse, die es
   * implementiert. Der Listener wird aufgerufen, wenn sich
   * das Name-Textfeld aendert. Das ist deswegen so umstaendlich
39 * (und nicht bspw. durch KeyListener), weil man eine
   * Namensaenderung auch mitbekommen will, wenn man einen Text
   * per Copy and Paste in das Textfeld fuegt...
42 */
  protected class DL implements DocumentListener {
   public void insertUpdate(DocumentEvent _) {
45  fname = fnameFeld.getText();
   }

48 public void removeUpdate(DocumentEvent _) {
    fname = fnameFeld.getText();
   }
51
   public void changedUpdate(DocumentEvent _) {
    fname = fnameFeld.getText();
54 }
  }

57 /**
   * Das Fenster, das erscheint, wenn ein Spieler ein neues
   * Spiel Starten will: Er wird aufgefordert seinen Namen einzugeben
60 * und die Schwierigkeitsstufe auszuwaehlen
   *
   * @param applet Laeuft das Spiel als Applet? Dann geht naemlich
63 * die Username-Ermittlung wieder nicht...
   */
  public TOptionenFenster(boolean applet) {
66 if (applet) {
    fname = "UnnamedPlayer";
   } else {
69  // sinnvoller Default-wert
    fname = System.getProperty("user.name");
   }
72
   final JDialog p = new JDialog(TSharedObjects.getMain().getHauptFenster(),
    "Neues Spiel...", true);
75 final Border margin = new EmptyBorder(8, 8, 8, 8);
   final JPanel mainPanel = new JPanel();

78 final ActionListener startButtonAL = new ActionListener() {
    public void actionPerformed(ActionEvent _) {
     p.dispose();
81   TSharedObjects.getAnzeige().neuesSpiel(fname, fschwierig);
    }
   };
84
   final JPanel panel1 = new JPanel();
   panel1.setBorder(new CompoundBorder(margin, new TitledBorder(null,
87  "Spielername", TitledBorder.LEFT, TitledBorder.TOP)));
   fnameFeld = new JTextField(fname, 20);
   fnameFeld.getDocument().addDocumentListener(new DL());
90 panel1.add(fnameFeld);
```

```
    final JPanel panel2 = new JPanel();
93  panel2.setBorder(new CompoundBorder(margin, new TitledBorder(null,
     "Schwierigkeitsgrad", TitledBorder.LEFT, TitledBorder.TOP)));
    final ButtonGroup bg = new ButtonGroup();
96  final JRadioButton rleicht = new JRadioButton("Leicht", false);
    final JRadioButton rmittel = new JRadioButton("Mittel", true);
    final JRadioButton rschwer = new JRadioButton("Schwer", false);
99
    final ActionListener radioButtonAL = new ActionListener() {
     public void actionPerformed(ActionEvent e) {
102   if (e.getActionCommand().toString().equals("0")) {
       fschwierig =
        TSpieler.Schwierigkeitsgrade.leicht;
105   } else if (e.getActionCommand().toString().equals("1")) {
       fschwierig =
        TSpieler.Schwierigkeitsgrade.normal;
108   } else if (e.getActionCommand().toString().equals("2")) {
       fschwierig =
        TSpieler.Schwierigkeitsgrade.schwer;
111   }
     }
    };
114
    rleicht.addActionListener(radioButtonAL);
    rleicht.setActionCommand("0");
117 rmittel.addActionListener(radioButtonAL);
    rmittel.setActionCommand("1");
    rschwer.addActionListener(radioButtonAL);
120 rschwer.setActionCommand("2");
    bg.add(rleicht);
    bg.add(rmittel);
123 bg.add(rschwer);
    panel2.add(rleicht);
    panel2.add(rmittel);
126 panel2.add(rschwer);

    final JPanel panel3 = new JPanel();
129 panel3.setBorder(margin);
    final JButton startButton = new JButton("...und␣los!");
    startButton.addActionListener(startButtonAL);
132
    panel3.add(startButton);

135 mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.Y_AXIS));
    mainPanel.add(panel1);
    mainPanel.add(panel2);
138 mainPanel.add(panel3);
    p.add(mainPanel);

141 p.pack();
    p.setResizable(false);
    p.setVisible(true);
144 p.repaint();
  }
}
```

Listing 18: TPaar.java

```java
package de.rccc.java.witchcraft;

/**
 * Generisches Paar fuer zwei Werte
 */
public class TPaar<A, B> {
 // Getter und Setter koennen wir uns hier sparen
 A eins;
 B zwei;

 /**
  * Konstruktor
  * @param a Wird in Feld 'eins' gefuellt
  * @param b Wird in Feld 'zwei' gefuellt
  */
 TPaar(A a, B b) {
  this.eins = a;
  this.zwei = b;
 }
}
```

Listing 19: TPartikel1.java

```java
package de.rccc.java.witchcraft;

import java.awt.Color;

/**
 * Partikel, der einen Funken darstellt
 */
public class TPartikel1 extends TObjekt implements IPartikel {
 /**
  * Konstruktor
  *
  * @param koord Der Koordinatenvektor
  * @param dim Der Groessenvektor
  * @param geschw Der Geschwindigkeitsvektor
  */
 TPartikel1(TVektor koord, TVektor dim, TVektor geschw) {
  super(koord, dim, geschw);
 }

 /**
  * Zeichnet das Bild an seinen Koordinaten auf der Zeichenflaeche g
  *
  * @param g Die Zeichenflaeche, auf der das Objekt gezeichnet werden soll
  */
 public void zeichne(java.awt.Graphics g) {
  g.setColor(Color.YELLOW);
  g.fillOval((int)fkoord.x, (int)fkoord.y, (int)fdim.x, (int)fdim.y);
 }
}
```

Listing 20: TPartikel2.java

```java
package de.rccc.java.witchcraft;

import java.awt.Color;
```

```
/**
 * Stellt einen Explosions-Partikel dar, der durch
 * mehrere gefuellte Kreise mit jeweils kleinerem Radius
 * und abgestuften Farben dargestellt wird
 */
public class TPartikel2 extends TObjekt implements IPartikel {
 int foffset = 1;

 /**
  * Konstruktor
  *
  * @param koord Der Koordinatenvektor
  * @param dim Der Groessenvektor
  * @param geschw Der Geschwindigkeitsvektor
  * @param offset Um wieviel sollen die inneren Kreise jeweils
  * verschoben sein
  */
 TPartikel2(TVektor koord, TVektor dim, TVektor geschw, int offset) {
  super(koord, dim, geschw);
  this.foffset = offset;
 }

 /**
  * Zeichnet das Bild an seinen Koordinaten auf der Zeichenflaeche g
  * Die Farben sind auf den Default-Grau-Hintergrund, der im Niedrig-
  * Detail-Modus verwendet wird, eingestellt, um wenigstens den
  * Anschein von zusammenpassender Grafik zu erwecken
  *
  * @param g Die Zeichenflaeche, auf der das Objekt gezeichnet werden soll
  */
 public void zeichne(java.awt.Graphics g) {
  int x = (int)fkoord.x;
  int y = (int)fkoord.y;
  int w = (int)fdim.x;
  int h = (int)fdim.y;
  g.setColor(new Color(91, 72, 61));
  g.fillOval(x, y, w, h);
  x += foffset; y += foffset; w -= foffset*2; h -= foffset*2;
  g.setColor(new Color(117, 79, 53));
  g.fillOval(x, y, w, h);
  x += foffset; y += foffset; w -= foffset*2; h -= foffset*2;
  g.setColor(new Color(149, 98, 50));
  g.fillOval(x, y, w, h);
  x += foffset; y += foffset; w -= foffset*2; h -= foffset*2;
  g.setColor(new Color(182, 131, 52));
  g.fillOval(x, y, w, h);
  x += foffset; y += foffset; w -= foffset*2; h -= foffset*2;
  g.setColor(new Color(233, 207, 70));
  g.fillOval(x, y, w, h);
 }
}
```

Listing 21: TPartikel3.java

```
package de.rccc.java.witchcraft;

import java.awt.Color;

/**
```

```
 * Stellt einen Funken dar, der aus dem Besenende sprueht
 */
public class TPartikel3 extends TObjekt implements IPartikel {
 /**
  * Konstruktor
  *
  * @param koord Der Koordinatenvektor
  * @param geschw Der Geschwindigkeitsvektor
  */
 TPartikel3(TVektor koord, TVektor geschw) {
  super(koord, new TVektor(10,10), geschw);
 }

 /**
  * Zeichnet das Bild an seinen Koordinaten auf der Zeichenflaeche g
  *
  * @param g Die Zeichenflaeche, auf der das Objekt gezeichnet werden soll
  */
 public void zeichne(java.awt.Graphics g) {
  int x = (int)fkoord.x;
  int y = (int)fkoord.y;

  g.setColor(new Color(132, 156, 172));
  g.fillRect(x + 4, y + 1, 3, 9);
  g.fillRect(x + 1, y + 4, 9, 3);
  g.setColor(new Color(255, 255, 255));
  g.fillRect(x + 6, y, 1, 11);
  g.fillRect(x, y + 6, 11, 1);
 }
}
```

Listing 22: TPartikelRauch.java

```
package de.rccc.java.witchcraft;

import java.awt.Color;

/**
 * Stellt ein Rauchwoelkchen dar, das von einer Rakete verursacht wird
 */
public class TPartikelRauch extends TObjekt implements IPartikel {
 /**
  * Konstruktor
  *
  * @param koord Der Koordinatenvektor
  * @param geschw Der Geschwindigkeitsvektor
  */
 TPartikelRauch(TVektor koord, TVektor geschw) {
  super(koord, new TVektor(8, 8), geschw);
 }

 /**
  * Zeichnet das Bild an seinen Koordinaten auf der Zeichenflaeche g
  *
  * @param g Die Zeichenflaeche, auf der das Objekt gezeichnet werden soll
  */
 public void zeichne(java.awt.Graphics g) {
  int x = (int)fkoord.x;
  int y = (int)fkoord.y;
```

```
27    g.setColor(new Color(89, 89, 89));
      g.fillOval(x-3, y-3, 7, 7);
      g.setColor(new Color(121, 121, 121));
30    g.fillOval(x, y, 2, 2);
    }
  }
```

Listing 23: TPartikelVerwaltung.java

```
  package de.rccc.java.witchcraft;

3 import java.util.*;
  import java.awt.Font;

6 /**
   * Partikel sind Objekte, die keine Kollision haben,
   * aber eine TTL und auf denen die Schwerkraft einwirkt
9  */
  public class TPartikelVerwaltung {
    /**
12   * Enum aller möglichen Partikelarten.
     * Diese enthalten auch die jeweiligen Gravitation-
     * und Reibungseinfluesse, die die Partikel erfahren sollen.
15   * Diese Werte gehoeren nicht in die Partikel, weil ein Partikel
     * nicht ueber sich selbst entscheiden koennen soll, wie er sich
     * bewegt.
18   */
    static public enum Partikel {
      /**
21     * NIL - Not in List. Das ist ein Fehlwert
       */
      NIL (0, 0),
24    /**
       * Funken Art 1. Kleine gelbe Punkte
       */
27    Funken1 (0.13, 0.990),
      /**
       * Funken Art 2. Kreuze. Der "Beseneffekt"
30     */
      Funken2 (0.13, 0.990),
      /**
33     * Explosionspartikel
       */
      Explosion (0.13, 0.990),
36    /**
       * Frosch. Ein Feind wurde zum Frosch verzaubert
       */
39    Frosch (0.13, 0.990),
      /**
       * Rauch, von Raketen verursacht - dieser steigt nach oben
42     * mit linearer Geschwindigkeit
       */
      Rauch (-0.15, 1),
45    /**
       * Score (die Punkte, die hochsteigen, wenn man einen
       * Gegner abgeschossen hat)
48     */
      Score (-0.05, 1);
```

```
51    /**
       * Die Reibung die der Partikel erfaehrt, bremst ihn ab
       */
54    private final double freibung;

      /**
57     * Die Gravitation die der Partikel erfaehrt, laesst
       * ihn nach unten fallen
       */
60    private final double fgravitation;

      /**
63     * Liest die Reibung fuer den Partikel
       */
      public double getReibung() {
66     return freibung;
      }

69    /**
       * Liest die Gravition fuer den Partikel
       */
72    public double getGravitation() {
       return fgravitation;
      }
75
      /**
       * Konstruktor
78     * @param reibung Welche Reibung soll der Partikel bei der
       * Bewegung erfahren
       * @param gravitation Welche Graviation soll der Partikel
81     * bei der Bewegung erfahren
       */
      Partikel(double reibung, double gravitation) {
84     freibung = reibung;
       fgravitation = gravitation;
      }
87   };

     /**
90    * Liste der Partikel. Jeder besteht aus einem Tripel: IPartikel o
      * und int i wobei o das entsprechende Objekt ist, und i die
      * Anzahl der Ticks, die das Objekt noch zu leben hat, sowie Partikel
93    * der Enum-Wert, der fuer das Verhalten (Flugbahn) zustaendig ist
      */
     List<TTripel<IPartikel, Integer, Partikel>> fpartikel =
96    new LinkedList<TTripel<IPartikel, Integer, Partikel>>();

     /**
99    * Sollen BildeObjekte oder programmierte Partikel verwendet werden?
      */
     private boolean fdetail = false;
102
     /**
      * Konstruktor der PartikelVerwaltung
105   */
     TPartikelVerwaltung() {
     }
108
     /**
```

```
     * Methode zum Starten eines neuen Effektes, der keine initiale
111  * Richtung mitbekommen soll
     *
     * @param p Welcher Effekt soll gestartet werden
114  * @param pos An welcher Stelle soll der Effekt gestartet werden
     */
    public void startEffekt(Partikel p, TVektor pos) {
117  startEffekt(p, pos, new TVektor(0, 0));
    }

120 /**
     * Methode zum Starten eines neuen Effektes, der einen String-
     * Parameter braucht - zB die Punkte, die erscheinen, wenn man
123  * einen Gegner abgeschossen hat
     *
     * @param p Welcher Effekt soll gestartet werden - momentan
126  * redunant, aber koennte auch noch mehr Effekte geben, die
     * einen String-Parameter brauchen, als nur 'Score'
     * @param pos An welcher Stelle soll der Effekt gestartet werden
129  * @param text Der Text-Parameter
     */
    public void startEffekt(Partikel p, TVektor pos, String text) {
132  // Score
     if (p == Partikel.Score) {
      TVektor richtung = new TVektor(0, -2);
135   Font font = new Font("Arial", Font.BOLD, (Integer.valueOf(text)/10)+10);
      fpartikel.add(new TTripel<IPartikel, Integer, Partikel>(
       new TText(new TVektor(pos), richtung, text, font),
138    40, Partikel.Score));
     }
    }
141

    /**
     * Methode zum Starten eines neuen Effektes. Effektiv werden
144  * entsprechende Partikel mit Zufallsrichtungen (in entsprechendem
     * Rahmen) an der gewuenschten Stelle erzeugt
     *
147  * @param p Welcher Effekt soll gestartet werden
     * @param pos An welcher Stelle soll der Effekt gestartet werden
     * @param inigeschw Startgeschwindigkeit und -richtung, die die Partikel
150  * bekommen sollen
     */
    public void startEffekt(Partikel p, TVektor pos, TVektor inigeschw) {
153  // gelbe Treffer-Funken
     if (p == Partikel.Funken1) {
      for (int i = 1; i <= 5; i++) {
156    TVektor richtung = new TVektor((TSharedObjects.rndDouble() * 6.0) -
        3.0 + inigeschw.x, (TSharedObjects.rndDouble()*2 - 2.5) +
        inigeschw.y);
159    if (!fdetail) {
        fpartikel.add(new TTripel<IPartikel, Integer, Partikel>(
         new TPartikel1(new TVektor(pos), new TVektor(3, 3), richtung),
162      30 + TSharedObjects.rndInt(40), Partikel.Funken1));
       } else {
        fpartikel.add(new TTripel<IPartikel, Integer, Partikel>(
165      new TBildObjekt(new TVektor(pos), new TVektor(7, 7), richtung,
         "partikel1"), 30 + TSharedObjects.rndInt(40), Partikel.Funken1));
       }
168   }
```

```
    }

    // grosse gelbe Explosions-Leuchtdinger
    if (p == Partikel.Explosion) {
     for (int i = 1; i <= 15; i++) {
      TVektor richtung = new TVektor((TSharedObjects.rndDouble() * 6.0) -
       3.0 + inigeschw.x, (TSharedObjects.rndDouble()*2 - 5.0) +
       inigeschw.y);
      if (!fdetail) {
       fpartikel.add(new TTripel<IPartikel, Integer, Partikel>(
        new TPartikel2(new TVektor(pos), new TVektor(25, 25), richtung, 2),
        30 + TSharedObjects.rndInt(40), Partikel.Explosion));
      } else {
       fpartikel.add(new TTripel<IPartikel, Integer, Partikel>(
        new TBildObjekt(new TVektor(pos), new TVektor(25, 25), richtung,
        "partikel2"), 30 + TSharedObjects.rndInt(40), Partikel.Explosion));
      }
     }
    }

    // blaue Besen-Funken
    if (p == Partikel.Funken2) {
     TVektor richtung = new TVektor((TSharedObjects.rndDouble() * 3.0) - 1.5,
      (TSharedObjects.rndDouble()*2 - 1.5));
     richtung.add(inigeschw);
     if (!fdetail) {
      fpartikel.add(new TTripel<IPartikel, Integer, Partikel>(
       new TPartikel3(new TVektor(pos), richtung),
       20 + TSharedObjects.rndInt(40), Partikel.Funken2));
     } else {
      fpartikel.add(new TTripel<IPartikel, Integer, Partikel>(
       new TBildObjekt(new TVektor(pos), null, richtung,
       "partikel3"), 20 + TSharedObjects.rndInt(40), Partikel.Funken2));
     }
    }

    // Frosch. Ein Feind wurde zum Frosch verzaubert
    if (p == Partikel.Frosch) {
     TVektor richtung = new TVektor((TSharedObjects.rndDouble() * 3.0) - 1.5,
       (TSharedObjects.rndDouble()*2 - 1.5));
     richtung.add(inigeschw);
     fpartikel.add(new TTripel<IPartikel, Integer, Partikel>(
      new TBildObjekt(new TVektor(pos), null, richtung, "frosch"),
      999, Partikel.Frosch));
    }

    // Rauch (steigt von Raketen auf)
    if (p == Partikel.Rauch) {
     if (!fdetail) {
      fpartikel.add(new TTripel<IPartikel, Integer, Partikel>(
       new TPartikelRauch(new TVektor(pos), inigeschw),
       20 + TSharedObjects.rndInt(40), Partikel.Rauch));
     } else {
      fpartikel.add(new TTripel<IPartikel, Integer, Partikel>(
       new TBildObjekt(new TVektor(pos), null, inigeschw,
       "rauch"), 20 + TSharedObjects.rndInt(40), Partikel.Rauch));
     }
    }
   }
```

```
228
    /**
     * Alle Effekte (Partikel) einen Tick weiterbewegen. Die TTL (Time To
231  * Live) der einzelnen Partikel wird heruntergesetzt, wenn dadurch
     * ein Partikel stirbt, wird er aus der Partikelliste entfernt
     */
234  public void update() {
      for (Iterator<TTripel<IPartikel, Integer, Partikel>> i =
       fpartikel.iterator(); i.hasNext();) {
237
       TTripel<IPartikel, Integer, Partikel> p = i.next();
       p.eins.gravitationAdd(p.drei.getReibung());
240    p.eins.gravitationMult(p.drei.getGravitation());
       p.eins.bewege();

243    p.zwei = p.zwei - 1;
       if (p.zwei <= 0) {
        i.remove();
246    } else if (p.eins.ausserhalbBildschirm()) {
        i.remove();
       }
249   }
    }

252 /**
     * Loescht alle Partikel
     */
255  public void reset() {
      fpartikel.clear();
    }
258
    /**
     * Aktuellen Stand des Partikelsystems zeichnen (d.h. alle Einzelpartikel
261  * an ihren aktuellen Positionen)
     * @param g Die Zeichenflaeche, auf der gezeichnet werden soll
     */
264  public void zeichne(java.awt.Graphics g) {
      for (TPaar<IPartikel, Integer> p: fpartikel) {
       p.eins.zeichne(g);
267   }
    }

270 /**
     * Setzt, ob neue Partikel BildObjekte oder programmierte Partikel
     * sein sollen
273  * @param d true fuer hohes, false fuer niedriges Detail
     */
    public void setHohesDetail(boolean d) {
276   fdetail = d;
    }
}
```

Listing 24: TRail.java

```
1  package de.rccc.java.witchcraft;

   /**
4   * Rail ist ein "instant Hit" Geschoss.
    * Das Ziel wird sofort getroffen.
```

```
 * Außerdem wird kein Bild gezeichnet, sondern eine Linie.
 */
public class TRail extends TGeschoss {
 /**
  * Von wo wurde es abgeschossen?
  */
 protected TVektor fquelle;

 /**
  * Wohin wurde geschossen?
  */
 protected TVektor fzielv;

 /**
  * Welches Objekt ist das Ziel?
  */
 protected TObjekt fziel;

 /**
  * Konstruktor
  *
  * @param quelle Woher kommt der der Schuss
  * @param ziel Welches Lebewesen ist das Ziel?
  * @param zielkoord Wo sind die Zielkoordinaten?
  * @param seite Welcher Seite gehört das Geschoss an?
  * @param waffe Mit welcher WAffe wurde geschossen?
  */
 TRail(TVektor quelle, TLebewesen ziel, TVektor zielkoord, int seite,
  TWaffe waffe) {

  super(zielkoord, new TVektor(), new TVektor(), seite, waffe);
  fquelle = quelle;
  fzielv = zielkoord;
  fziel  = ziel;
  // Instant hit
  fttl=10;
 }

 /**
  * Zeichnet den Rail an seinen Koordinaten auf der Zeichenflaeche g
  *
  * @param g Die Zeichenflaeche, auf der das Objekt gezeichnet werden soll
  */
 public void zeichne(java.awt.Graphics g) {
  // Hier NICHT: super.zeichne(g);
  g.setColor(java.awt.Color.BLUE);
  g.drawLine((int)fquelle.x,(int)fquelle.y,
   (int)fzielv.x,(int)fzielv.y);
 }

 /**
  * Ueberprueft, ob das Objekt mit einem Anderen kollidiert
  *
  * @param anderes Das andere Objekt, mit dem auf Kollision geprueft
  * werden soll
  * @return Besteht eine Kollision (ueberschneiden sich die Objekte)?
  */
 public boolean beruehrt(TLebewesen anderes) {
  // Nur EINMAL!
```

```
    if ((anderes == fziel) && (anderes.innerhalb(fzielv))) {
     fziel = null;
     return true;
    } else {
     return false;
    }
   }
}
```

Listing 25: TRakete.java

```
package de.rccc.java.witchcraft;

/**
 * Eine Rakete ist ein Selbstlenkgeschoss.
 * Die Rakete hat zwei Modi: DumbFire oder Verfolgung.
 * im DumbFire dreht sich die Rakete nach dem Start auf den Zielvektor,
 * und fliegt dann diese Richtung weiter. Im Verfolgermodus verfolgt
 * die Rakete ein Ziel, indem die sich immer in Zielrichtung dreht
 * und dann beschleunigt.
 */
public class TRakete extends TGeschoss {
 /**
  * Maximaler Drehwinkel pro Tick
  */
 private static final double MAX_DREHW = 0.15;

 /**
  * Maximale Beschleunigung pro Tick
  */
 private static final double BESCHL = 1;

 /**
  * Das Zielobjekt, auf das die Rakete zufliegt
  */
 protected TObjekt fziel = null;

 /**
  * Wie ist der Zielwinkel, auf dem die Rakete einschwenken soll,
  * wenn kein ZielObjekt da ist.
  */
 protected double fzielw;

 /**
  * Mitte der Rakete (wird fuer die Kollision verwendet)
  */
 protected TVektor fmitte;

 /**
  * Maixmalgeschwindigkeit
  */
 protected int fmaxgeschw;

 /**
  * Konstruktor fuer bekanntes Zielobjekt
  *
  * @param koord Startkoordinaten der Rakete
  * @param dim Groessenvektor
  * @param geschw Geschwindigkeitsvektor
```

```
     * @param seite Fuer welche Seite fliegt die Rakete
     * @param waffe Die Waffe, aus der die Rakete erzeugt wurde
51   * @param ausrichtung Anfaenglicher Drehwinkel
     * @param ziel Das Zielobjekt der Lenkrakete
     * @param zielw Der ZielVektor, der Rakete. ACHTUNG! ENTWEDER ziel
54   * ODER zielw! (Soll viele fast-identische Konstruktoren vermeiden)
     * @param maxgeschw Maximale Geschw. der Rakete
     */
57   TRakete(TVektor koord, TVektor dim, TVektor geschw, int seite,
     TWaffe waffe, double ausrichtung, TObjekt ziel,
     double zielw, int maxgeschw) {
60
     super(koord, dim, geschw, seite, waffe);
     fausrichtung = Math.toRadians(ausrichtung);
63   fziel  = ziel;
     fzielw = zielw;
     fmitte = new TVektor(fdim.x/2 , fdim.y/2);
66   fmaxgeschw = maxgeschw;
    }

69  /**
     * Die Rakete bekommt ein neues Ziel zugewiesen
     */
72  public void setZiel(TObjekt ziel) {
     fziel = ziel;
    }
75
    /**
     * Die Rakete wird auf das Ziel ausgerichtet, sofern eines
78   * vorhanden ist, anschliessend in diese Richtung bewegt
     */
    public void bewege() {
81   double winkelsoll;
     double winkeldiff;

84   if (fziel != null) {
      // Dazu den benötigen Vektor finden, der zum Ziel führt
      TVektor zielmitte = new TVektor(fziel.fdim.x/2, fziel.fdim.y/2);
87    TVektor ziel      = new TVektor(fziel.fkoord);
      ziel.add(zielmitte);

90    TVektor raketenmitte = fkoord.newAdd(fmitte);

      TVektor zielv = ziel.newSub(raketenmitte);
93    TVektor aktiv = new TVektor(Math.sin(fausrichtung),
      Math.cos(fausrichtung));
      winkelsoll = aktiv.winkel(zielv);
96    winkeldiff =  winkelsoll;
     } else {
      // Der Winkel ist der wirkliche
99    winkelsoll = fzielw;
      winkeldiff = (fausrichtung - winkelsoll) * -1;
     }
102
     // Nun hätten wir also den benötigten Winkel. Aber die Rakete kann
     // sich nicht so schnell drehen
105  // Die Ausrichtung der Rakete korrigieren
     if (winkeldiff > MAX_DREHW) {
      winkeldiff = MAX_DREHW;
```

```
108     } else if (winkeldiff < -MAX_DREHW) {
          winkeldiff = -MAX_DREHW;
        }
111     fausrichtung += winkeldiff;

        // winkeldiff nicht neu rechnen. Könnte einen guten Effekt bringen
114
        // Zu dem Winkel einen Vektor generieren
        // Und jetzt darf die Rakete Gas geben
117     double faktor=1;

        if (winkeldiff > Math.PI/2) {
120       // Wenn die sich vom Ziel abwendet, dann kein Gas geben
          faktor = 0;
        } else if (winkeldiff > Math.PI/4) {
123       // Hier anteilig Gas geben
          faktor = 1 / ((Math.PI/4) * winkeldiff);
        } else {
126       // Vollgas
          faktor = 1;
        }
129     if (fziel != null) {
          // Falls das Ziel zerstört wird
          fzielw = fausrichtung;
132     }

        TVektor dazu = new TVektor(Math.sin(fausrichtung)*BESCHL*faktor,
135       Math.cos(fausrichtung)*BESCHL*faktor);
        fgeschw.add(dazu);

138     if (fgeschw.laenge() > fmaxgeschw) {
          // Objekt ist zu schnell. Abbremsen
          // Aber nur langsam
141       double setgeschw = fmaxgeschw;/*fgeschw.laenge()-BESCHL;
          if (setgeschw<MAX_GESCHW) {
            setgeschw=MAX_GESCHW;
144       }*/
          fgeschw.setlaenge(setgeschw);
        }
147
        // Der Winkel ist der wirkliche
        super.bewege();
150
        // Und nu die Richtigen Bilder organisieren
        berechneBildangaben();
153
        // Rauch
        if (TSharedObjects.rndInt(5) == 0) {
156       TVektor akt = new TVektor(fgeschw);
          akt.mult(-1);
          // Und nu, so lang machen, wie die Rakete maxGeschw ist
159       akt.mult(fmaxgeschw/akt.laenge());
          akt.add(fgeschw);

162       TSharedObjects.getPartikelVerwaltung().startEffekt(
            TPartikelVerwaltung.Partikel.Rauch, fkoord,
            akt);
165     }
      }
```

```
168   /**
       * Ist das Objekt ausserhalb des Bildschirms UND hat kein Ziel?
       */
171   public boolean ausserhalbBildschirm() {
       return (fziel == null) && super.ausserhalbBildschirm();
      }
174 }
```

Listing 26: TSavegame.java

```
   package de.rccc.java.witchcraft;

3  import java.io.*;
   import java.util.*;
   import org.w3c.dom.*;
6  import org.xml.sax.*;
   import org.xml.sax.helpers.*;
   import javax.xml.transform.*;
9  import javax.xml.parsers.*;
   import javax.xml.transform.stream.*;
   import javax.xml.transform.sax.*;
12 import java.awt.*;
   import java.awt.event.*;
   import javax.swing.*;
15 import javax.swing.event.*;
   import java.text.DateFormat;

18 /**
    * Klasse die statische Methoden zum Laden und Speichern des
    * aktuellen Spielstandes enthaelt
21  */
   public class TSavegame {
    /**
24   * Da die Java-Interne Liste ironischerweise das Interface
     * ListModel nicht implementiert, benoetigen wir jetzt unsere
     * eigene Version einer Liste, die auch mit JList
27   * zusammenarbeitet.
     */
    private static class TGenerischeListe implements ListModel {
30   /**
      * Die eigentliche Liste der Elemente
      */
33   private java.util.List<Object> felemente =
      new ArrayList<Object>();

36   /**
      * Nur fuer die Erfuellung des Interface benoetigt
      */
39   public void addListDataListener(ListDataListener _) {}

     /**
42    * Nur fuer die Erfuellung des Interface benoetigt
      */
     public void removeListDataListener(ListDataListener _) {}
45
     /**
      * Fuegt ein Objekt zur Liste hinzu
48    */
```

```
  public void add(Object o) {
   felemente.add(o);
  }

  /**
   * Liefert das Objekt an der Stelle Index
   */
  public Object getElementAt(int index) {
   return felemente.get(index);
  }

  /**
   * Liefert die Groesse der Liste
   */
  public int getSize() {
   return felemente.size();
  }
 }

 /**
  * Gibt das Verzeichnis an, in dem die Savegames abgelegt
  * werden sollen
  */
 private static String home =
  System.getProperty("user.home") + "/.witchcraft/save";

 /**
  * Methode, die ein Fenster zur Verfuegung stellt, mit dem man
  * eines der vorhandenen Savegames zum Laden auswaehlen kann
  */
 public static void savegameWaehlen() {
  final Map<String, Integer> eintraege = new HashMap<String, Integer>();
  final TGenerischeListe strings = new TGenerischeListe();
  final JDialog p = new JDialog(TSharedObjects.getMain().getHauptFenster(),
   "Bitte Savegame wählen", true);
  TSharedObjects.getAnzeige().updateEnable(false);
  TSharedObjects.getAnzeige().pause();

  // Liste von Eintraegen bauen
  try {
   int i = 1;
   while (new File(home + "/save" + i + ".xml").exists()) {
    File f = new File(home + "/save" + i + ".xml");
    Document datei = DocumentBuilderFactory.newInstance()
     .newDocumentBuilder().parse(f);
    NodeList nds = datei.getChildNodes().item(0).getChildNodes();

    String levelnummer = null;
    String spielername = null;
    String punkte = null;

    for (int j = 0; j < nds.getLength(); j++) {
     String nodeName = nds.item(j).getNodeName();
     String inhalt = nds.item(j).getTextContent();
     if (nodeName.equals("levelnummer")) {
      levelnummer = inhalt;
     } else if (nodeName.equals("spielername")) {
      spielername = inhalt;
     } else if (nodeName.equals("punkte")) {
```

```
108         punkte = inhalt;
            }
           }
111
           String angezeigt = "" + i + ":␣␣␣" + "Spieler:␣" +
            spielername + "␣␣␣Level:␣" + levelnummer +
114         "␣␣(" + punkte + "␣Punkte)␣␣--␣" +
            DateFormat.getDateTimeInstance()
             .format(new Date(f.lastModified()));
117         strings.add(angezeigt);
            eintraege.put(angezeigt, i);
            i++;
120        }
       } catch (Exception e) {
        TSharedObjects.getAnzeige().setMeldung("Savegame␣laden␣" +
123      "fehlgeschlagen");
        TSharedObjects.getAnzeige().updateEnable(true);
        p.dispose();
126    }

       // Fenster und so weiter bauen
129    final JList liste = new JList(strings);
       liste.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
       final JScrollPane scrollPane = new JScrollPane(liste);
132    scrollPane.setMinimumSize(new Dimension(650, 450));
       scrollPane.setSize(new Dimension(650, 450));
       scrollPane.setPreferredSize(new Dimension(650, 450));
135    final JButton ok = new JButton("OK");

       ok.addActionListener(new ActionListener() {
138     public void actionPerformed(ActionEvent _) {
         Integer ausgewaehlt = eintraege.get(liste.getSelectedValue());
         if (ausgewaehlt != null) {
141       p.dispose();
          TSharedObjects.getAnzeige().lade(eintraege.get(
           liste.getSelectedValue()));
144      }
        }
       });
147
       p.add(scrollPane, BorderLayout.CENTER);
       p.add(ok, BorderLayout.SOUTH);
150
       p.pack();
       p.setVisible(true);
153    TSharedObjects.getAnzeige().updateEnable(true);
      }

156   /**
       * Gibt an, ob mindestens ein Savegame vorhanden ist
       * @return true, falls mindestens ein Savegame gefunden wurde
159    * false falls keines gefunden wurde
       */
      public static boolean mindEinSavegameVorhanden() {
162    File f = new File(home + "/save1.xml");
       return f.exists();
      }
165
      /**
```

```
     * Lädt und aktiviert ein Waffe aus dem Savegame
168  * @param n Die Node, aus der die Waffe erzeugt werden soll
     * @param spieler Das Spielerobjekt, das die Waffe erhalten soll
     */
171  public static void ladeWaffe(Node n, TSpieler spieler) throws Exception {
     // in die Liste eintauchen
     NodeList nds = n.getChildNodes();
174  Node a;

     String id = null;
177  int abgefeuert = 0;
     int treffer = 0;

180  for (int i = 0; i < nds.getLength(); i++) {
      a = nds.item(i);

183   if (a.getNodeName().equals("ID")) {
       id = a.getTextContent();
      } else if (a.getNodeName().equals("Abeschossen")) {
186    abgefeuert = Integer.parseInt(a.getTextContent());
      } else if (a.getNodeName().equals("Treffer")) {
       treffer = Integer.parseInt(a.getTextContent());
189   }
     }

192  if (id == null) {
      throw new Exception("Fehler beim Levelladen: Waffen-ID nicht gesetzt");
     } else if (TWaffe.getWaffe(id) == null) {
195   throw new Exception("Fehler beim Levelladen: Waffen " + id +
       " existiert nicht");
     }
198  spieler.addWaffe(id);
     TWaffe.getWaffe(id).setStatistik(abgefeuert, treffer);
    }
201

    /**
     * Laedt das Savegame mit der Nummer
204  * @param nummer Die Nummer des Savegames, das geladen werden soll
     * @param spieler Das Spielerobjekt, dessen Werte aus dem Savegame geladen
     * werden sollen
207  * @return Das Levelobjekt, an dessen Stelle das neue Level geladen
     * werden soll
     */
210  public static TLevel lade(int nummer, TSpieler spieler) throws Exception {
     String dateiname = home + "/save" + nummer + ".xml";
     System.out.println("Lade " + dateiname + "...");
213

     Map<String, String> werte = new HashMap<String, String>();

216  Document datei = DocumentBuilderFactory.newInstance()
      .newDocumentBuilder().parse(new File(dateiname));
     NodeList nds = datei.getChildNodes();
219

     if ((nds.getLength() != 1) || (nds.item(0).getNodeName() !=
      "savegame")) {
222   throw new Exception("Fehlerhaftes Savegame: " + datei);
     } else {
      nds = nds.item(0).getChildNodes();
225   for (int i = 0; i < nds.getLength(); i++) {
```

```
      if (nds.item(i).getNodeName().equals("waffe")) {
       //waffen.add(nds.item(i).getTextContent());
228    ladeWaffe(nds.item(i), spieler);
      } else {
       werte.put(nds.item(i).getNodeName(),
231     nds.item(i).getTextContent());
      }
     }
234  }

     if ((werte.get("spielername") == null) ||
237   (werte.get("spielerx") == null) ||
      (werte.get("spielery") == null) ||
      (werte.get("levelnummer") == null) ||
240   (werte.get("levelfortschritt") == null) ||
      (werte.get("punkte") == null) ||
      (werte.get("ausgewähltewaffe") == null) ||
243   (werte.get("leben") == null) ||
      (werte.get("lebenmax") == null) ||
      (werte.get("schwierigkeit") == null)) {
246
      throw new Exception("Fehlerhaftes Savegame: " + datei);
     }
249
     spieler.setKoord(new TVektor(Double.valueOf(werte.get("spielerx")),
      Double.valueOf(werte.get("spielery"))));
252  spieler.setName(werte.get("spielername"));
     spieler.setAktiveWaffe(Integer.valueOf(werte.get("ausgewähltewaffe")));
     spieler.setScore(Integer.valueOf(werte.get("punkte")));
255  spieler.setLeben(Double.valueOf(werte.get("leben")));
     spieler.setLebenMax(Double.valueOf(werte.get("lebenmax")));
     spieler.setSchwierigkeit(
258   TSpieler.Schwierigkeitsgrade.valueOf(werte.get("schwierigkeit")));

     return new TLevel(Integer.valueOf(werte.get("levelnummer")),
261   Integer.valueOf(werte.get("levelfortschritt")), true);
    }

264 /**
     * Speichert das Spiel. Es wird hier SAX zum Schreiben verwendet,
     * weil das hiermit mehr "straightforward" geht.
267  * @param spieler Das Spielerobjekt, dessen Score usw.
     * gespeichert werden soll
     * @param level Das Levelobjekt, dessen Fortschritt gespeichert
270  * werden soll
     */
    public static void speichere(TSpieler spieler,
273  TLevel level) throws Exception {

     // Savegame-Verzeichnis anlegen, falls nicht vorhanden
276  File f = new File(home);
     if (!f.exists()) {
      f.mkdirs();
279  }

     // naechste freie Savegame-Nummer herausfinden
282  int i = 1;
     while (new File(home + "/save" + i + ".xml").exists()) {
      i++;
```

```
285     }
        String datei = home + "/save" + i + ".xml";
288
        // was soll in das Savegame geschrieben werden?
        java.util.List<TPaar<String, String>> werte =
291      new ArrayList<TPaar<String, String>>();
        werte.add(new TPaar<String, String>("spielername",
         spieler.getName()));
294     werte.add(new TPaar<String, String>("spielerx",
         Double.toString(spieler.getKoord().x)));
        werte.add(new TPaar<String, String>("spielery",
297      Double.toString(spieler.getKoord().y)));
        werte.add(new TPaar<String, String>("levelnummer",
         Integer.toString(level.getLevelNummer())));
300     werte.add(new TPaar<String, String>("levelfortschritt",
         Integer.toString(level.getGeloeschteSegmente())));
        werte.add(new TPaar<String, String>("punkte",
303      Integer.toString(spieler.getScore())));
        werte.add(new TPaar<String, String>("ausgewähltewaffe",
         Integer.toString(spieler.getAktiveWaffe())));
306     werte.add(new TPaar<String, String>("leben",
         Double.toString(spieler.getLeben())));
        werte.add(new TPaar<String, String>("lebenmax",
309      Double.toString(spieler.getLebenMax())));
        werte.add(new TPaar<String, String>("schwierigkeit",
         spieler.getSchwierigkeit().toString()));
312
        PrintWriter out = new PrintWriter(new FileOutputStream(datei));
        StreamResult streamResult = new StreamResult(out);
315     SAXTransformerFactory tf = (SAXTransformerFactory)SAXTransformerFactory
         .newInstance();
        TransformerHandler hd = tf.newTransformerHandler();
318     Transformer serializer = hd.getTransformer();
        // Uebler Hack fuer seltsames Verhalten unter Windows mit unbekanntem
        // Ursprung. Siehe Technische Dokumentation fuer weiter Erlaeuterung
321     if (System.getProperty("os.name").equals("Windows␣XP")) {
         serializer.setOutputProperty(OutputKeys.ENCODING, "ISO-8859-15");
        } else {
324      serializer.setOutputProperty(OutputKeys.ENCODING, "UTF-8");
        }
        serializer.setOutputProperty(OutputKeys.INDENT, "yes");
327     hd.setResult(streamResult);
        hd.startDocument();
        hd.startElement("", "", "savegame", null);
330
        for(TPaar<String, String> wert: werte) {
         hd.startElement("", "", wert.eins, null);
333      hd.characters(wert.zwei.toCharArray(), 0, wert.zwei.length());
         hd.endElement("", "", wert.eins);
        }
336     // Die Waffen können wir nicht so elegant speichern
        for (TWaffe w: spieler.getWaffenListe()) {
         String id = w.getID();
339      String abgeschossen = "" + w.getStatAbgefeuert();
         String treffer = "" + w.getStatTreffer();
         hd.startElement("", "", "waffe", null);
342      hd.startElement("", "", "ID", null);
         hd.characters(id.toCharArray(), 0, id.length());
```

```
    hd.endElement("", "", "ID");
345    hd.startElement("", "", "abgeschossen", null);
    hd.characters(abgeschossen.toCharArray(), 0, abgeschossen.length());
    hd.endElement("", "", "abgeschossen");
348    hd.startElement("", "", "treffer", null);
    hd.characters(treffer.toCharArray(), 0, treffer.length());
    hd.endElement("", "", "treffer");
351    hd.endElement("", "", "waffe");
   }

354  hd.endElement("", "", "savegame");
   hd.endDocument();
  }
357 }
```

Listing 27: TSharedObjects.java

```
package de.rccc.java.witchcraft;

3 import java.util.Map;
import java.util.HashMap;
import java.util.Random;
6
/**
 * TSharedObject enthält Referenzen zu allen Objekten, die von vielen
9 * Klassen benötigt werden
 */
public class TSharedObjects {
12 /**
  * statische Liste aller Bilder
  */
15 protected static Map<String, TAnimation> bildliste =
  new HashMap<String, TAnimation>();

18 /**
  * Sound-Objekt
  */
21 protected static TSound fsound = null;

 /**
24  * Partikel-Verwaltung
  */
 protected static TPartikelVerwaltung fpverw = null;
27
 /**
  * Spiel-Anzeige
30  */
 protected static TAnzeige fanzeige = null;

33 /**
  * die Haupt-Spielklasse registiert sich fuer alle erreichbar,
  * damit zB Menueeintraege disabled werden koennen
36  */
 protected static Main fmain = null;

39 /**
  * Zufallsgenerator
  */
42 protected static Random frandom = new Random();
```

```
    /**
45   * Gibt an, ob gerade Musik spielt (und ob beim Spielstart
     * Musik direkt anfangen soll oder nicht)
     */
48  protected static boolean fmusik = false;

    /**
51   * Fuer alle zugaenglich: Die Breite des Spielfensters
     */
    public final static int FENSTER_BREITE = 640;
54
    /**
     * Fuer alle zugaenglich: Die Hoehe des Spielfensters
57   */
    public final static int FENSTER_HOEHE = 480;

60  /**
     * Liefert ein Zufalls-Int
     */
63  public static int rndInt(int r) {
     return frandom.nextInt(r);
    }
66
    /**
     * Liefert ein Zufalls-Double
69   */
    public static double rndDouble() {
     return frandom.nextDouble();
72  }

    /**
75   * Fügt ein neues Bild in die Bilderlsite an
     */
    static public void addBild(String id, TAnimation bild) {
78   bildliste.put(id, bild);
    }

81  /**
     * Gibt eine Kopie eine Bildes aus der Bilderliste zurück
     * Beim Kopieren der TAnimation werden die Animationsframes
84   * nicht kopiert, sondern referenziert. Alle Objekte, die eigene
     * Animationsframes haben (also Gegner etc. - nicht Geschosse, da hier
     * die Animationsframes die Drehphasen darstellen) verwenden daher
87   * diese Methode anstelle von getBild()
     *
     * @param bild Das Bild
90   */
    public static TAnimation getNewBild(String bild) {
     return (bildInListe(bild) ? new TAnimation(bildliste.get(bild)) : null);
93  }

    /**
96   * Gibt eine Refernz auf ein Bild aus der Bilderliste zurück
     *
     * @param bild Das Bild
99   */
    public static TAnimation getBild(String bild) {
     return bildliste.get(bild);
```

```
102  }

     /**
105   * Liefert die Information darueber, ob ein Bild in der Bildliste
      * vorhanden ist
      * @param bild Das Bild, das in der Liste gesucht wird
108   * @return Ist das Bild in der Liste vorhanden?
      */
     public static boolean bildInListe(String bild) {
111   return (bildliste != null) && (bildliste.get(bild) != null);
     }

114  /**
      * Setzt die Partikelverwaltung
      */
117  public static void setPartikelVerwaltung(TPartikelVerwaltung pverw) {
      fpverw = pverw;
     }
120
     /**
      * Gibt die aktuell Verwendete Partikelverwaltung zurück
123   * (besser als die Aufrufe durchzureichen wie beim Sound)
      */
     public static TPartikelVerwaltung getPartikelVerwaltung() {
126   return fpverw;
     }

129  /**
      * Setzt die Anzeige
      * @param az Die zu setzende Anzeige
132   */
     public static void setAnzeige(TAnzeige az) {
      fanzeige = az;
135  }

     /**
138   * Liefert die Anzeige zurueck
      */
     public static TAnzeige getAnzeige() {
141   return fanzeige;
     }

144  /**
      * Setzt die Haupt-Spielklasse - von allen zugreifbar, damit
      * z.B. Menueintraege disabled werden koennen
147   */
     public static void setMain(Main main) {
      fmain = main;
150  }

     /**
153   * Liefert die Haupt-Spielklasse
      */
     public static Main getMain() {
156   return fmain;
     }

159  /**
      * Liefert, ob Musik gerade laeuft (beim Spielstart: ob direkt
```

```
   * beim Spielstart die Musik anfangen soll)
   */
  public static boolean getMusik() {
   return fmusik;
  }

  /**
   * Schaltet die Musik ein oder aus und aendert alles notwendige
   */
  public static void setMusik(boolean musik) {
   fmusik = musik;
   if (musik) {
    TSound.playMusik("MUSIK");
    fmain.aendereMenuItem("optionen_musik", "Musik abschalten");
   } else {
    TSound.stoppe("MUSIK");
    fmain.aendereMenuItem("optionen_musik", "Musik anschalten");
   }
  }

  /**
   * Methode, die das Spiel beendet sofern moeglich,
   * andernfalls zumindest das Fenster schliesst
   */
  public static void endGame() {
   try {
    System.exit(0);
   } catch (java.security.AccessControlException e) {
    // Wir koennen im Applet das Programm nicht beenden -
    // aber wir koennen das Fenster schliessen :)
    if (fmusik) {
     setMusik(false);
     TSound.stoppe("MUSIK");
    }

    fmain.getHauptFenster().dispose();
    System.out.println("Kann Programm nicht beenden.");
    fanzeige.pause();
   }
  }
}
```

Listing 28: TSound.java

```
package de.rccc.java.witchcraft;

import java.util.*;
import java.io.File;
import java.io.IOException;
import java.applet.*;

/**
 * Soundklasse des Spiels.
 *
 * Idee:
 * Die Sound-daten sind statisch, die werden nur einmal gebraucht.
 * Wenn nun ein Sound abgespielt wird, wird eine neue Instanz dieses
 * Objektes erzeugt, und als Thread gestartet. Im Thread passiert
 * das Abspielen.
```

```
 *
 * Da das AudoClip aber nur eine Instanz ist, und wenn man den Sound
 * öfter Abspielt, das immer wieder abgebrochen und von vorne gespielt
 * wird, müssen wir tricksen:
 * - kopieren geht nicht :(  (Da Audioclip ein Interface ist)
 * - daher n-AudioClips mit dem gleichen Sound erzeugen, und dann beim
 * Abspielen dieses Round-Robin abspielen...
 */
public class TSound extends Applet implements Runnable{
 private static final int MEHRFACH = 4;

 /**
  * Eine Liste der Sounds mit den IDs.
  * Die Sounds aber als Array im "paar": Da wir sonst immer nur
  * EINE instanz hätten, würde bei zweimalen abspielen der Sound
  * abgebrochen, und neu gestartet. Daher einen "vorrat" an Sounds
  * anlegen, die parralell spielen können. "eins" in "paar" zeigt
  * auf das nächste zu nutzende Soundobjekt.
  */
 private static Map<String, TPaar<Integer, AudioClip[]>> sounds =
  new HashMap<String, TPaar<Integer, AudioClip[]>>();

 /**
  * Der Sound kann abgeschaltet werden.
  * Wenn es z.B. Probleme (Weil geht nicht) gibt
  */
 private static boolean faktiv = true;

 /**
  * Ist das Musik. (soll geloopt werdern)
  */
 private boolean fmusik = false;

 /**
  * Die Auswahl des abzuspielenden Audioclip passiert im Create,
  * NICHT im Thread!  Das so gemacht, weil ich nicht weiss, was
  * passiert, wenn mehrere Threads gleichzeitig im TPaar.eins
  * rumfummeln
  */
 private AudioClip fsound;

 /**
  * Konstruktor
  *
  * @param sound Der Sound, der abgespielt werden soll
  */
 TSound(String sound, boolean musik) {
  // Das NICHT in den Thread! Keine Ahnung was passiert, wenn
  // zwei Threads gleichzeitig die Zahl vom Paar ändern wollen!
  TPaar<Integer, AudioClip[]> paar = sounds.get(sound);
  Integer x = paar.eins;
  fsound = paar.zwei[x];
  if (fsound == null) {
   // BUG! BUG! BUG!
   // Das war wohl eine Musikstück, das zweimal abgespielt wurde.
   fsound = paar.zwei[0];
  }
  x++;
  if (x >= MEHRFACH) {
```

```
75      x = 0;
        }
        paar.eins = x;
78      fmusik = musik;
      }

81    /**
       * Fügt einen Sound in die Liste hinzu
       *
84     * @param id Die ID des Sounds
       * @param datei die Datei
       * @param musik ist das eine Musik-Date?
87     */
      public static void addSound(String id, String datei,
       boolean musik) throws Exception {
90
       java.net.URL url = TSound.class.getResource("/sounds/" + datei);

93     if (url == null) {
        throw new Exception("Sounddatei \"" + datei + "\" nicht vorhanden.");
       }
96
       AudioClip[] sndarr = new AudioClip[MEHRFACH];

99     if (musik) {
        // Musik wird nur einmal abgespielt, außerdem frisst
        // die Musik speicher
102     sndarr[0] = Applet.newAudioClip(url);
        for (int x = 1; x < MEHRFACH; x++) {
         sndarr[x] = null;
105     }

       } else {
108     for (int x = 0; x < MEHRFACH; x++) {
         sndarr[x] = Applet.newAudioClip(url);
        }
111    }
       TPaar<Integer, AudioClip[]> paar =
        new TPaar<Integer, AudioClip[]>(0, sndarr);
114    sounds.put(id, paar);
      }

117   /**
       * Hier wird der Sound abgespielt.
       *
120    * Das ist als Thread-Methode gemacht
       */
      public void run() {
123    if (faktiv) {
        try {
         if (fmusik) {
126       fsound.loop();
          while (true) {
           Thread.sleep(100000);
129       }
         } else {
          fsound.play();
132       // Leider können wir nicht abfragen, ob der Sound
          // schon zuende ist, daher raten
```

```
      Thread.sleep(1000);
135     }
      } catch (Exception e) {
       System.out.println("Soundproblem: " + e + " bei " + fsound);
138    e.printStackTrace();
       faktiv = false;
       System.out.println("Sound wird deaktiviert.");
141    }
     }
    }
144
    /**
     * Einen Sound abspielen
147  *
     * @param sound Welcher Sound abgespielt werden soll
     */
150  static public void play(String sound) {
     // Null kann vorkommen und ist auch legitim (z.B. haben nicht
     // alle Waffen alle Sounds)
153  if (sound != null) {
      TSound snd = new TSound(sound, false);
      Thread soundthread = new Thread(snd);
156   soundthread.start();
     }
    }
159
    /**
     * Musik abspielen
162  *
     * @param sound Welche Musik abgespielt werden soll
     */
165  static public void playMusik(String sound) {
     TSound snd = new TSound(sound, true);
     Thread soundthread = new Thread(snd);
168  soundthread.start();
    }

171 /**
     * Stoppt alle Sounds der ID
     */
174  static public void stoppe(String sound) {
     TPaar<Integer, AudioClip[]> paar = sounds.get(sound);
     AudioClip snd;
177  for (int x = 0; x < MEHRFACH; x++) {
      snd = paar.zwei[x];
      if (snd != null) {
180    // Die Musik hat nur einen Arrayeintrag!
       snd.stop();
      }
183  }
    }

186 /**
     * Plausibilitaetspruefung:
     * Prüft, ob ein Sound in der Liste vorhanden ist.
189  *
     * @param sound der Sound, der geprüft werden soll
     * @return true, wenn Sound in Liste
192  */
```

```
  static public boolean inListe(String sound) {
   return sounds.get(sound) != null;
  }
}
```

Listing 29: TSpieler.java

```
1  package de.rccc.java.witchcraft;

   import java.util.*;
4  import java.awt.*;

   /**
7   * Dieses hier ist das Objekt des Spielers.
    * Daher werden hier viele Sonderkekse gebacken.
    */
10 public class TSpieler extends TLebewesen {
    /**
     * Mögliche bewegungen des Spielers. Die Eintraege einzeln
13   * zu kommentieren ist reichlich sinnlos...
     */
    public static enum Bewegung {
16   oben,
     unten,
     rechts,
19   links
    };

22  /**
     * Die Schwierigkeitsgrade, die hier unterstüzt werden
     */
25  public static enum Schwierigkeitsgrade {
     /**
      * leicht (250 leben, 150 magie)
28    */
     leicht,
     /**
31    * normal (100 leben, 100 magie)
      */
     normal,
34   /**
      * schwer (50 leben, 75 magie)
      */
37   schwer
    };

40  /**
     * Magie des Lebewesens
     */
43  protected double fmagie;

    /**
46   * Maximale Magie des Lebewesens
     */
    protected double fmaxmagie;
49
    /**
     * Taste nach oben gedrückt
52   */
```

```
    protected boolean foben = false;

55  /**
     * Taste nach unten gedrückt
     */
58  protected boolean funten = false;

    /**
61   * Taste nach rechts gedrückt
     */
    protected boolean frechts = false;
64
    /**
     * Taste nach links gedrückt
67   */
    protected boolean flinks = false;

70  /**
     * Maximales Inkrement fuer beide Richtungen (Bewegungsweite)
     */
73  protected int fdmax = 5;

    /**
76   * Wie viele Punkte hat der Spieler
     */
    protected int fscore;
79
    /**
     * Koordinaten fuer den fraktenauswurf, relativ zum Spieler
82   */
    final protected static TVektor fraktenauswurf = new TVektor(50, 100);

85  /**
     * Startkoordinaten fuer den Rail-Strahl, relativ zum Spieler
     */
88  final protected static TVektor frailauswurf = new TVektor(50, 50);

    /**
91   * Koordinaten fuer den Auswurf der Dumb-Fire Waffen
     * (Frosch-Zauber), relativ zum Spieler
     */
94  final protected static TVektor dfauswurf = new TVektor(80, 30);

    /**
97   * Ende des Besens, für die Partikel, relativ zum Spieler
     */
    final protected static TVektor fbesenende = new TVektor( 0, 70);
100
    /**
     * Merkvariablen für das  Dauefeuer: Ist Das Feuer aktiv?
103  */
    protected boolean ffaueraktiv = false;
    /**
106  * Merkvariablen für das  Dauefeuer: Dieses Lebewesen ist unser Ziel
     */
    protected TLebewesen fziel;
109 /**
     * Merkvariablen für das  Dauefeuer: Dieses sind unsere Zielkoordinaten
     */
```

```
112  protected TVektor fzielv = new TVektor();

     /**
115   * Eine Liste aller Waffen, die der Spieler hat
      */
     protected java.util.List<TWaffe> fwaffenliste = new ArrayList<TWaffe>();
118
     /**
      * Aktuell ausgewaehlte Waffe
121   */
     protected int faktWaffe = -1;

124  /**
      * Rate of Fire
      * Zähler. Wenn 0, dann darf das nächste Mal gefeuert werden.
127   * Wird bei bewege() runtergezählt
      */
     protected int frof = 0;
130
     /**
      * Zeigt die aktive Waffe an
133   */
     protected TText fhudText1 = null;

136  /**
      * Zeigt die Score des Spielers an
      */
139  protected TText fhudText2 = null;

     /**
142   * Der Name des Spielers (fuer eine Highscore)
      */
     protected String fname = "UnnamedPlayer";
145
     /**
      * Gibt den aktuellen Schwierigkeitsgrad an
148   */
     protected Schwierigkeitsgrade fschwierigkeit = Schwierigkeitsgrade.normal;

151  /**
      * Countdown für die Statistik am Levelende
      */
154  protected int fLevelEnde;

     /**
157   * Array der Texte, die am Levelende angezeigt werden
      */
     protected java.util.List<TText> fLevelEndeTextListe =
160   new ArrayList<TText>();

     /**
163   * Konstruktor
      *
      * @param schwer Der Schwirigkeitsgrad
166   */
     public TSpieler(Schwierigkeitsgrade schwer) {

169   // Lebenspunkte sind temporär, das wird bei "setSchwierigkeit" gesetzt
      super(new TVektor(1,1), null, new TVektor(0,0), "SPIELER", 10, 0);
```

```
172     fgeschw.set(0,0);

        setSchwierigkeit(schwer);
175
        Font hudFont = new Font("Arial", Font.BOLD, 13);
        fhudText1 = new TText(new TVektor(5, 470), null, hudFont);
178     fhudText2 = new TText(new TVektor(500, 470), "Punkte: 0", hudFont);
        fLevelEnde = 0;
      }
181
      /**
       * Bewege den Spieler einen Schritt
184    */
      public void bewege() {
       super.bewege();
187
       // Starten wir ein paar Funken
       if (TSharedObjects.rndInt(3) == 0) {
190     TSharedObjects.getPartikelVerwaltung().startEffekt(
          TPartikelVerwaltung.Partikel.Funken2, fbesenende.newAdd(fkoord),
          new TVektor(-3, 0));
193    }

       if (fmagie < fmaxmagie) {
196     fmagie += 1;
       }

199    if (fkoord.x<0) {
        fkoord.x = 0;
        fru.x = fdim.x;
202    } else if ((fkoord.x+fdim.x) > TSharedObjects.FENSTER_BREITE) {
        fkoord.x = TSharedObjects.FENSTER_BREITE - fdim.x;
        fru.x = TSharedObjects.FENSTER_BREITE;
205    }
       if (fkoord.y<0) {
        fkoord.y = 0;
208     fru.y = fdim.y;
       } else if ((fkoord.y+fdim.y) > TSharedObjects.FENSTER_HOEHE) {
        fkoord.y = TSharedObjects.FENSTER_HOEHE - fdim.y;
211     fru.y = TSharedObjects.FENSTER_HOEHE;
       }

214    if (frof > 0) {
        frof -= 1;
       }
217
       // Und jetzt feuern
       if (ffaueraktiv && (frof<=0)) {
220     feuer();
       }

223    // Partikel-Effekt des Besens
       if (TSharedObjects.rndInt(3) == 0) {
        TSharedObjects.getPartikelVerwaltung().startEffekt(
226       TPartikelVerwaltung.Partikel.Funken2,
          fbesenende.newAdd(fkoord),
          new TVektor(-3,0));
229    }
```

```
    }

232 /**
     * Der Spieler will sich bewegen
     *
235  * @param bewegung wohin bewegen?
     * @param gedrueckt Ist die Taste gedrueckt
     */
238 public void bewegen(Bewegung bewegung,  boolean gedrueckt) {
     boolean xakt = false;
     boolean yakt = false;
241  if (bewegung == Bewegung.oben) {
      foben = gedrueckt;
      yakt = true;
244  } else if (bewegung == Bewegung.unten) {
      funten = gedrueckt;
      yakt = true;
247  } else if (bewegung == Bewegung.rechts) {
      frechts = gedrueckt;
      xakt = true;
250  } else if (bewegung == Bewegung.links) {
      flinks = gedrueckt;
      xakt = true;
253  }

     if (xakt) {
256   if (frechts == flinks) {
       // keine, oder beide gedrückt
       fgeschw.x = 0;
259   } else if (frechts) {
       fgeschw.x = fdmax;
      } else if (flinks) {
262    fgeschw.x = -fdmax;
      }
     }
265
     if (yakt) {
      if (foben == funten) {
268    // keine, oder beide gedrückt
       fgeschw.y = 0;
      } else if (foben) {
271    fgeschw.y = -fdmax;
      } else if (funten) {
       fgeschw.y = fdmax;
274   }
     }

277 }

    /**
280  * Die Maustaste wurde gedrückt.
     * Jetzt dafür sorgen, dass in "bewege" gefeuert wird
     */
283 public void clickaktiv() {
     ffaueraktiv = true;
    }
286
    /**
     * Die Maustaste wurde losgelassen.
```

```
289     * Feuern einstellen
        */
       public void clicknichtaktiv() {
292     ffaueraktiv = false;
       }

295    /**
        * Die Maus wurde bewegt.
        * Jetzt die Zielkoordianten ändern
298     */
       public void zielgeaendert(TLebewesen ziel, int x, int y) {
        fziel = ziel;
301     fzielv.set(x,y);
       }

304    /**
        * auf das Ziel feuern
        */
307    private void feuer() {
        if (faktWaffe >= fwaffenliste.size()) {
         System.out.println("Problem: Waffe ausgewählt die gibt es " +
310       "nicht. Aktiviere die letzte Waffe");
         faktWaffe = fwaffenliste.size() - 1;
        }
313
        // Keine Waffe ausgewaehlt? - passiert z.b., wenn im Level
        // vergessen wurde, den Spieler zu bewaffnen.
316     if (faktWaffe < 0) {
         return;
        }
319
        TWaffe aktw = fwaffenliste.get(faktWaffe);
        TAnzeige anzeige = TSharedObjects.getAnzeige();
322     // frof wurde schon geprüft (in bewege)
        if (fmagie>=aktw.getMagie()) {
         fmagie -= aktw.getMagie();
325      frof = aktw.getRof();

         // Prüfen, ob das aktuelle Ziel nicht schon tot ist
328      if ((fziel != null) && (fziel.getLeben()<0)) {
          fziel = null;
         }
331
         if (aktw.getWaffe() == TWaffe.Waffen.Dumbfire) {
          // Dumbfire Waffe. Einmal in die Richtung feuern
334
          // Zielvekoor errechnen. Das ist der Zielpunkt
          TVektor zielp = new TVektor(fzielv);
337       // Minus meinen Koordinaten
          zielp.sub(fkoord);
          // Minus den "auswurf"-koordinaten
340       zielp.sub(dfauswurf);
          // Geschwindigkeit achten
          zielp.setlaenge(aktw.getGeschw());
343       anzeige.addGeschoss(new TGeschoss(fkoord.newAdd(dfauswurf), null,
           zielp, fseite, aktw));
         } else if (aktw.getWaffe() == TWaffe.Waffen.Rail) {
346       // "Strahl" waffe
          anzeige.addGeschoss(new TRail(fkoord.newAdd(frailauswurf), fziel,
```

```
        new TVektor(fzielv), fseite, aktw));
    } else if (aktw.getWaffe() == TWaffe.Waffen.Rakete) {
      TRakete rak1;
      TRakete rak2;
      TVektor zielp = null;

      if (fziel == null) {
        // Raketenmodus "DumbFire"
        zielp = new TVektor(fzielv);
        zielp.sub(getRU());
        rak1 = new TRakete(fkoord.newAdd(fraktenauswurf),
          null, new TVektor(fgeschw.x, 2),
          fseite, fwaffenliste.get(faktWaffe), 60,
          null, TVektor.genormt.winkel(zielp),
          fwaffenliste.get(faktWaffe).getGeschw());
        rak2 = new TRakete(fkoord.newAdd(fraktenauswurf),
          null, new TVektor(fgeschw.x, 2),
          fseite, fwaffenliste.get(faktWaffe), 120,
          null, TVektor.genormt.winkel(zielp),
          fwaffenliste.get(faktWaffe).getGeschw());
      } else {
        rak1 = new TRakete(fkoord.newAdd(fraktenauswurf),
          null, new TVektor(fgeschw.x, 2),
          fseite, fwaffenliste.get(faktWaffe), 60, fziel, 0,
          fwaffenliste.get(faktWaffe).getGeschw());
        rak2 = new TRakete(fkoord.newAdd(fraktenauswurf),
          null, new TVektor(fgeschw.x, 2),
          fseite, fwaffenliste.get(faktWaffe), 120, fziel, 0,
          fwaffenliste.get(faktWaffe).getGeschw());
        fziel.addAbhaengige(rak1);
        fziel.addAbhaengige(rak2);
      }

      anzeige.addGeschoss(rak1);
      anzeige.addGeschoss(rak2);
    }
  }
}

/**
 * Es wurde ein Mausklick (rechte Maustaste) gemeldet.
 *
 * @param ziel Ggf. ein Lebewesen, das anvisiert wurde
 * @param x X-Koordinate des Klicks
 * @param y Y-Koordinate des Klicks
 */
public void clickrechts(TLebewesen ziel, int x, int y) {
  nextWaffe();
}

/**
 * Eine weitere Waffe zur Liste hinzufugen
 *
 * @param waffe ID der Waffe
 */
public void addWaffe(String waffe) throws Exception {
  TWaffe neu = TWaffe.getWaffe(waffe);

  if (neu == null) {
```

```
    throw new Exception("Unbekannte Waffe: \"" + waffe + "\"");
   }

   if (!fwaffenliste.contains(neu)) {
    fwaffenliste.add(neu);
    TSharedObjects.getAnzeige().setMeldung("Waffe erhalten: " + neu);
    setAktiveWaffe(fwaffenliste.size() - 1);
   }
  }

  /**
   * Alle Waffen, die der Spieler bis jetzt eingesammelt hat,
   * zuruecksetzen (zB bei einem neuen Levelstart)
   */
  public void leereWaffen() {
   fwaffenliste.clear();
  }

  /**
   * Die naechste verfuegbare Waffe auswaehlen
   */
  public void nextWaffe() {
   setAktiveWaffe(faktWaffe + 1);
  }

  /**
   * Setzt die aktive Waffe des Spielers
   */
  public void setAktiveWaffe(int w) {
   faktWaffe = w;
   if (faktWaffe >= fwaffenliste.size()) {
    if (fwaffenliste.size() == 0) {
     // Spieler hat keine Waffe
     faktWaffe = -1;
    } else {
     faktWaffe = 0;
    }
   }

   fhudText1.setText("aktive Waffe: " + fwaffenliste.get(faktWaffe));
  }

  /**
   * Den Spieler und die zum Spieler gehoerigen Sachen (zB den Magiebalken)
   * zeichnen
   * @param g Die Zeichenflaeche, auf der gezeichnet werden soll
   */
  public void zeichne(java.awt.Graphics g) {
   super.zeichne(g);

   // Magie-Balken
   g.setColor(java.awt.Color.BLACK);
   g.fillRect(0, TSharedObjects.FENSTER_HOEHE-10,
    TSharedObjects.FENSTER_BREITE, 6);
   g.setColor(java.awt.Color.BLUE);
   g.fillRect(2, TSharedObjects.FENSTER_HOEHE-8,
    (int)((TSharedObjects.FENSTER_BREITE-2)*(fmagie/fmaxmagie)), 2);
   fhudText1.zeichne(g);
   fhudText2.zeichne(g);
```

```
466   }

      /**
469    * Zeichnet die Statistik am Ende des Levels
       */
      protected void zeichneStats(java.awt.Graphics g) {
472    if (fLevelEnde == 0) {
        Font font = new Font("Arial", Font.BOLD, 13);
        fLevelEndeTextListe.add(new TText(new TVektor(10, 50),
475      "Waffe", font));
        fLevelEndeTextListe.add(new TText(new TVektor(200, 50),
         "Abgefeuert", font));
478     fLevelEndeTextListe.add(new TText(new TVektor(290, 50),
         "Treffer", font));
        fLevelEndeTextListe.add(new TText(new TVektor(360, 50),
481      "Sonderpunkte", font));
       } else if (fLevelEnde % 10 == 0) {
        Font font = new Font("Arial", Font.BOLD, 13);
484     int i = fLevelEnde / 10 -1;

        // Die nächste Waffe anzeigen
487     if (i < fwaffenliste.size()) {
         TWaffe w = fwaffenliste.get(i);
         fLevelEndeTextListe.add(new TText(new TVektor(10, 80+i*20),
490       w.toString(), font));
         fLevelEndeTextListe.add(new TText(new TVektor(200, 80+i*20),
          ""+w.getStatAbgefeuert(), font));
493      fLevelEndeTextListe.add(new TText(new TVektor(290, 80+i*20),
          ""+w.getStatTreffer(), font));

496      // Sonderpunkte ausrechnen
         // Wenn nicht genügend abgefeuert wurden, kann man
         // nichts vergeben
499      double trefferrate = (double)
          w.getStatTreffer() / w.getStatAbgefeuert();
         trefferrate *= 100;
502      if (w.getStatAbgefeuert() < 20) {
          fLevelEndeTextListe.add(new TText(new TVektor(360, 80+i*20),
           "Nicht␣genügend␣Schüsse", font));
505      } else if (trefferrate < 50) {
          fLevelEndeTextListe.add(new TText(new TVektor(360, 80+i*20),
           "Zu␣niedrige␣Trefferrate:␣" + (int)trefferrate + "%", font));
508      } else if (trefferrate < 80) {
          fLevelEndeTextListe.add(new TText(new TVektor(360, 80+i*20),
           "Trefferrate:␣" + (int)trefferrate +
511        "%␣␣␣Gut:␣+1000␣Score", font));
          addScore(1000);
          TSound.play("GELD");
514      } else {
          fLevelEndeTextListe.add(new TText(new TVektor(360, 80+i*20),
           "Trefferrate:␣" + (int)trefferrate +
517        "%␣␣␣Sehr␣Gut!␣+2000␣Score", font));
          addScore(2000);
          TSound.play("GELD");
520      }
        }
       }
523    fLevelEnde++;
```

```
    // Und nun zeichnen
526  for (TText i:fLevelEndeTextListe) {
      i.zeichne(g);
     }
529 }

    /**
532  * Die Statusmeldungen auf den Anfangszustand setzen
     * (beim naechsten Zeichnen ein Neu-Aufbauen der Liste erzwingen)
     */
535 public void statsRuecksetzen() {
     fLevelEndeTextListe.clear();
     fLevelEnde = 0;
538 }

    /**
541  * Addiert die Punktezahl des Spielers
     */
    public void addScore(int score) {
544  fscore += score;
     fhudText2.setText("Punkte: " + fscore);
    }
547

    /**
     * Setzt die Punkte des Spielers
550  */
    public void setScore(int score) {
     fscore = score;
553  addScore(0); // Zum zeichnen
    }

556 /**
     * Wird aufgerufen, wenn der Spieler ein Item aufsammelt
     */
559 public void addItem(TItem item) throws Exception{
     int faktor;

562  switch (item.getItem()) {
      case Heilung:
       if (fschwierigkeit == Schwierigkeitsgrade.leicht) {
565     faktor = 2;
       } else {
        faktor = 1;
568    }
       treffer(-item.getiParam() * faktor);
       break;
571   case Waffe:
       addWaffe(item.getsParam());
       break;
574  }
    }

577 /**
     * Setzt den Namen des Spielers
     */
580 public void setName(String name) {
     fname =  name;
    }
583
```

```
    /**
     * Liefert die Anzahl der Punkte, die der Spieler gemacht hat
586  */
    public int getScore() {
     return fscore;
589 }

    /**
592  * Liefert die aktuell ausgewaehlte Waffe des Spielers
     */
    public int getAktiveWaffe() {
595  return faktWaffe;
    }

598 /**
     * Liefert den Namen des Spielers zurueck
     */
601 public String getName() {
     return fname;
    }
604
    /**
     * Liefert den Schwierigkeitsgrad zurueck
607  */
    public Schwierigkeitsgrade getSchwierigkeit() {
     return fschwierigkeit;
610 }

    /**
613  * Setzt den Schwierigkeitsgrad
     */
    public void setSchwierigkeit(Schwierigkeitsgrade s) {
616  fschwierigkeit = s;
     switch (s) {
      case leicht:
619    flebenmax = 250;
       fmaxmagie = 150;
       break;
622    case normal:
       flebenmax = 100;
       fmaxmagie = 100;
625    break;
      case schwer:
       flebenmax = 50;
628    fmaxmagie = 75;
       break;
     }
631
     // Das wird nur bei einem neuen Spiel aufgerufen
     fmagie = fmaxmagie;
634  flebenspunkte = flebenmax;

     // Den Balken neu berechnen
637  treffer(0);
    }

640 /**
     * Liefert die Liste der Waffen, die der Spieler zur Verfuegung hat
     */
```

```
643   public java.util.List<TWaffe> getWaffenListe() {
       return fwaffenliste;
      }
646
      /**
       * Leer die Waffenliste
649    */
      public void leereWaffenListe() {
       fwaffenliste.clear();
652   }

    }
```

Listing 30: TText.java

```
    package de.rccc.java.witchcraft;

3   import java.awt.Color;
    import java.awt.Font;

6   /**
     * Textobjekt, das auch fuer ein automatisches Ausblenden des
     * Textes nach einer bestimmten Anzahl von Darstellungen (Ticks)
9    * sorgen kann
     */
    public class TText extends TObjekt implements IPartikel {
12   /**
      * Der anzuzeigende String
      */
15   protected String ftext;

     /**
18    * Wie soll der String bei der Ausgabe formatiert werden
      */
     protected Font ffont;
21
     /**
      * Wie oft soll noch gezeichnet werden?
24    */
     protected int ftimeout = -1;

27   /**
      * Konstruktor
      *
30    * @param koord Der Koordinatenvektor
      * @param text Der anzuzeigende Text
      * @param font Wie soll der String bei der Ausgabe formatiert werden
33    * @param timeout Wie viele Ticks soll der String angezeigt werden
      */
     TText(TVektor koord, String text, Font font, int timeout) {
36    this(koord, text, font);

      ftimeout = timeout;
39   }

     /**
42    * Konstruktor
      *
      * @param koord Der Koordinatenvektor
```

```
45   * @param text Der anzuzeigende Text
     * @param font Wie soll der String bei der Ausgabe formatiert werden
     */
48   TText(TVektor koord, String text, Font font) {
      super(koord, null, null);

51    ftext = text;
      ffont = font;
     }
54
     /**
      * Konstruktor
57    *
      * @param koord Der Koordinatenvektor
      * @param geschw Der Richtungsvektor
60    * @param text Der anzuzeigende Text
      * @param font Wie soll der String bei der Ausgabe formatiert werden
      */
63   TText(TVektor koord, TVektor geschw, String text, Font font) {
      super(koord, new TVektor(1, 1), geschw);

66    ftext = text;
      ffont = font;
     }
69
     /**
      * Zeichnet das Bild an seinen Koordinaten auf der Zeichenflaeche g
72    *
      * @param g Die Zeichenflaeche, auf der das Objekt gezeichnet werden soll
      */
75   public void zeichne(java.awt.Graphics g) {
      if (ftimeout > 0) {
       ftimeout--;
78    }

      if ((ftimeout != 0) && (ftext != null)) {
81     g.setFont(ffont);
       // Erst ein Bisschen versetzt schwarz zeichnen, dann mittig in
       // gelb. Gibt einen netten Umrandungseffekt
84     g.setColor(Color.BLACK);
       g.drawString(ftext, (int)fkoord.x + 1, (int)fkoord.y + 1);
       g.drawString(ftext, (int)fkoord.x - 1, (int)fkoord.y - 1);
87     g.drawString(ftext, (int)fkoord.x + 1, (int)fkoord.y - 1);
       g.drawString(ftext, (int)fkoord.x - 1, (int)fkoord.y + 1);
       g.setColor(Color.YELLOW);
90     g.drawString(ftext, (int)fkoord.x, (int)fkoord.y);
      }
     }
93
     /**
      * Setzt den Text, der dauerhaft angezeigt werden soll
96    */
     public void setText(String text) {
      ftext = text;
99    ftimeout = -1;
     }

102  /**
      * Setzt einen Text, der nach 80 Ticks verschwindet
```

```
   */
105  public void setTimeoutText(String text) {
    ftext = text;
    ftimeout = 80;
108  }

   /**
111   * Liefert den eingestellten Text zurueck
    */
   public String getText() {
114    return ftext;
   }
}
```

Listing 31: TTripel.java

```
package de.rccc.java.witchcraft;

3 /**
   * Generisches Tripel fuer drei Werte
   */
6 public class TTripel<A, B, C> extends TPaar<A, B> {
  C drei;

9  TTripel(A a, B b, C c) {
    super(a, b);
    this.drei = c;
12  }
}
```

Listing 32: TVektor.java

```
1 package de.rccc.java.witchcraft;

  import javax.xml.parsers.*;
4 import org.w3c.dom.*;

  /**
7  * Klasse fuer 2-dimensionale Vektoren. x- und y-Felder werden
   * von den nutzenden Klassen so oft gebraucht, dass auf separate
   * Get- und Set-Methoden verzichtet wurde. Die Klasse enthaelt
10 * eine Reihe von Vektoroperationen, die fuer das Spiel nuetzlich
   * sind.
   */
13 public class TVektor {
   /**
    * X-Komponente des Vektors
16  */
   public double x;

19 /**
    * Y-Komponente des Vektors
    */
22 public double y;

   /**
25  * Normierter Vektor e2.
    * Wird zur Winkelberechnung benötigt
    */
```

```
28  static public TVektor genormt = new TVektor(0, 1);

    /**
31   * Defaultkonstruktor: Vektor wird mit zwei Nullwerten initialisiert
     */
    TVektor() {
34   this(0., 0.);
    }

37  /**
     * Konstruktor: Vektor mit zwei Zahlenwerten initialisieren
     *
40   * @param x Die x-Komponente als double-Wert
     * @param y Die y-Komponente als double-Wert
     */
43  TVektor(double x, double y) {
     this.x = x;
     this.y = y;
46  }

    /**
49   * Konstruktor: Vektor wird mit zwei Int-Werten initialisiert
     *
     * @param x Die x-Komponente als int-Wert
52   * @param y Die y-Komponente als int-Wert
     */
    TVektor(int x, int y) {
55   this.x = (double)x;
     this.y = (double)y;
    }
58
    /**
61   * Kopierkonstruktor
     *
     * @param v Der Vektor, der kopiert werden soll
     */
64  TVektor(TVektor v) {
     this.x = v.x;
     this.y = v.y;
67  }

    /**
70   * Setzen der x,y Koordinaten
     *
     * @param x x-Korrdinate
73   * @param y y-Korrdinate
     */
    public void set(double x, double y) {
76   this.x = x;
     this.y = y;
    }
79
    /**
82   * Setzt den Vektor auf die Koordinaten des anderen Vektors.
     * Gemacht, falls man nicht kopieren, bzw. zuweisen will
     *
     * @param vektor der andere Vektor
85   */
    public void set(TVektor vektor) {
```

```
    this.x = vektor.x;
88  this.y = vektor.y;
    }

91  /**
     * Addiere einen Vektor
     *
94   * @param v Der zu addierende Vektor
     */
    public void add(TVektor v) {
97   this.x += v.x;
     this.y += v.y;
    }
100
    /**
     * Addiere einen Vektor und liefere das Ergebnis
103  * als neuen Vektor zurueck
     *
     * @param v Der zu addierende Vektor
106  * @return Der durch die Addition entstandene Vektor
     */
    public TVektor newAdd(TVektor v) {
109  return new TVektor(this.x + v.x, this.y + v.y);
    }

112 /**
     * Subtrahiere einen Vektor
     *
115  * @param v Der zu subtrahierende Vektor
     */
    public void sub(TVektor v) {
118  this.x -= v.x;
     this.y -= v.y;
    }
121
    /**
     * Substrahiert einen Vektor und liefere das Ergebnis
124  * als neuen Vektor zurueck
     *
     * @param v Der zu subtrahierende Vektor
127  * @return Der durch die Addition entstandene Vektor
     */
    public TVektor newSub(TVektor v) {
130  return new TVektor(this.x - v.x, this.y - v.y);
    }

133 /**
     * Multipliziert einen Skalar mit dem Vektor
     *
136  * @param s Der Skalar, mit dem multipliziert werden soll
     */
    public void mult(double s) {
139  this.x *= s;
     this.y *= s;
    }
142
    /**
     * Berechne das Skalarprodukt zweier Vektoren
145  *
```

```
     * @param v Der Vektor, mit dem das SP berechnet werden soll
     * @return Das berechnete Skalarprodukt
148  */
    public double skalarprodukt(TVektor v) {
     return (x * v.x) + (y * v.y);
151 }

    /**
154  * Berechne die Laenge eines Vektors
     *
     * @return Die Laenge des Vektors
157  */
    public double laenge() {
     return Math.sqrt(x*x + y*y);
160 }

    /**
163  * Berechne den Winkel zwischen zwei Vektoren, in Bogenmass
     *
     * @param v Der Vektor, zwischen dem der Winkel berechnet
166  * werden soll
     * @return Der Winkel der Vektoren in Bogenmass
     */
169 public double winkel(TVektor v) {
     double ret = skalarprodukt(v) / (laenge() * v.laenge());
     if (ret>1) {
172   // Manchmal ist ret ein kleines bisschen größer als 1.
      // Und das mag acos nicht
      ret=1;
175  };
     ret = Math.acos(ret);

178  // Jetzt rausfinden, wie die Vektoren zueinander liegen.
     // Dazu das Kreuzprodukt berechnen. Da aber die x und y
     // Achse garantiert 0 sind, reicht uns die z Achse
181  double kreuz_z = x * v.y - y * v.x;

     if (kreuz_z > 0) {
184   ret *= -1;
     }

187  return ret;
    }

190 /**
     * Set den Vektor auf eine bestimmte Laenge setzen
     *
193  * @param laenge Die neue Laenge des Vektors
     */
    public void setlaenge(double laenge) {
196  double aktlaenge = laenge();
     double faktor = laenge / aktlaenge;
     this.x *= faktor;
199  this.y *= faktor;
    }

202 /**
     * Tauscht die beiden Kompanjenten aus
     */
```

```
205   public void swap() {
       double a = x;
       x = y;
208    y = a;
      }

211   /**
       * Textuelle Ausgabe des Strings (Debug-Ausgabe)
       *
214    * @return Formatierter String, der X- und Y-Komponenten enthaelt
       */
      public String toString() {
217    return "[" + x + ",␣" + y + "]";
      }

220   /**
       * Koordinaten aus einem XML-Node auslesen.
       * Besonderheit: Ändert die eigenden Daten, KEIN neues Objekt
223    *
       * @param n Der zu lesende Node
       */
226   public void leseAusXmlNode(Node n) {
       // in die Bilder eintauchen
       NodeList nds = n.getChildNodes();
229    // wie aktuell ;)
       Node a;

232    for (int i = 0; i < nds.getLength(); i++) {
         a = nds.item(i);
        if (a.getNodeName().equals("x")) {
235      x = Integer.parseInt(a.getTextContent());
       } else if (a.getNodeName().equals("y")) {
         y = Integer.parseInt(a.getTextContent());
238    }
      }
     }
241 }
```

Listing 33: TWaffe.java

```
1  package de.rccc.java.witchcraft;

   import javax.xml.parsers.*;
4  import org.w3c.dom.*;
   import java.util.*;
   import java.awt.Font;
7  import java.awt.*;

   /**
10  * Waffendefinitionen, die im Spiel verwendet werden.
    * Hier sind nur die "Definitionen" - die sind pro Waffe einamlig.
    * Die Bildschirmobjekte werden in TGeschoss gespeichert.
13  */
   public class TWaffe {
    /**
16   * Die zur Verfügung stehenden Behandlungen.
     * Alle Geschosse haben eine dieser Arten, die die Behandlung
     * (anzeige, bewegung, etc) vorgibt.
19   */
```

```
static public enum Waffen {
 /**
22  * Not in List. Fehlwert.
  */
 NIL,
25 /**
  * Raketenart.
  * Ein Objekt, das sich nach dem ausstoß in die Zielrichtung
28  * dreht, und dann dahinfliegt, bzw. das ein Ziel verfolgt
  */
 Rakete,
31 /**
  * "Instant hit" Waffe.
  * Das Ziel wird sofort getroffen, und es wird ein strahl dahingemalt
34  */
 Rail,
 /**
37  * DumbFire.
  * Standardtyp. Die Waffe fliegt in eine Richtung. Ohne sich
  * zu drehen, etc.
40  */
 Dumbfire
};
43
/**
 * Liste der verfügbaren Waffen (static)
46 */
protected static Map<String, TWaffe> fwaffenliste =
 new HashMap<String, TWaffe>();
49
/**
 *  Die benoetigte Magie
52 */
protected int fmagie = 0;

55 /**
 * Den Schaden, den die Waffe ausrichtet
 */
58 protected int fschaden = 0;

/**
61 * Wie viele Ticks soll der Bildschirm wackeln bei
 * Abschuss eines Gegners mit dieser Waffe
 */
64 protected int fbeben = 0;

/**
67 * Um welche Art handelt es sich
 */
protected Waffen fwaffe;
70
/**
 * Rate of fire
73 */
protected int frof;

76 /**
 * Wie heisst diese Waffe. Nur bei spielerauswählbaren waffen wichtig
 */
```

```
79  protected String fbez;

    /**
82   * ID der Waffe - diese wird auch ins Savegame geschrieben
     */
    protected String fid;
85
    /**
     * Mit welchem Bild soll das dargestellt werden?
88   */
    protected String fbild;

91  /**
     * Mit welcher Geschwindig fliegt das?
     */
94  protected int fgeschw;

    /**
97   * Welcher Sound soll beim abfeuern dieser Waffe gespielt werden?
     */
    protected String fstartsound;
100
    /**
     * Welcher Sound soll beim auftreffen (Einschlag) dieser Waffe
103  * gespielt werden?
     */
    protected String fhitsound;
106
    /**
     * Welcher Sound soll beim töten mit dieser Waffe gespielt werden?
109  */
    protected String ftotsound;

112 /**
     * Welche Partikel sollen beim Treffer mit dieser Waffe erzeugt werden?
     */
115 protected TPartikelVerwaltung.Partikel fpartikeltreff =
     TPartikelVerwaltung.Partikel.NIL;

118 /**
     * Welche Partikel sollen bei Zerstörung mit dieser Waffe erzeugt werden?
     */
121 protected TPartikelVerwaltung.Partikel fpartikeltot =
     TPartikelVerwaltung.Partikel.NIL;

124 /**
     * Statistik: Wie oft wurde diese Waffe abgefeuert
     */
127 protected int fstatAbgefeuert = 0;

    /**
130  * Statistik: Wie oft wurde mit dieser Waffe getroffen
     */
    protected int fstatTreffer = 0;
133
    /**
     * Statistik: Die Anzahl der abgefeuerten eröhen
136  */
    public void statIncAbgefeuert() {
```

```
    fstatAbgefeuert++;
139 }

    /**
142  * Statistik: Die Anzahl der getroffenen erhöhen
     */
    public void statIncTreffer() {
145  fstatTreffer++;
    }

148 /**
     * Statistik: Gib die Anzahl der abgefeuerten Schüsse zurück
     */
151 public int getStatAbgefeuert() {
     return fstatAbgefeuert;
    }
154
    /**
     * Statistik: Gib die Anzahl der Treffer zurück
157  */
    public int getStatTreffer() {
     return fstatTreffer;
160 }

    /**
163  * Statistik: Reset
     */
    public static void resetStatistik() {
166  for (String i : fwaffenliste.keySet()) {
      TWaffe akt = fwaffenliste.get(i);
      akt.fstatAbgefeuert = 0;
169   akt.fstatTreffer = 0;
     }
    }
172
    /**
     * Statisitk setzen (für das Laden des Levels)
175  *
     * @param statAbgefeuert Wie viele Schuesse wurden mit dieser
     * Waffe abgefeuert
178  * @param statTreffer Und wie viele davon waren Treffer
     */
    public void setStatistik(int statAbgefeuert, int statTreffer) {
181  fstatAbgefeuert = statAbgefeuert;
     fstatTreffer = statTreffer;
    }
184
    /**
     * Holt eine Waffe zu der ID
187  *
     * @param id Die ID der zu suchenden Waffe
     */
190 public static TWaffe getWaffe(String id) {
     return fwaffenliste.get(id);
    }
193
    /**
     * Fügt eine Waffe in der Waffenliste hinzu
196  *
```

```
     * @param id Die ID der zuzufuegenden Waffe
     * @param waffe Die Waffe, die der Liste zugefuegt werden soll
199  */
    public static void addWaffe(String id, TWaffe waffe) {
     fwaffenliste.put(id, waffe);
202 }

    /**
205  * Konstruktor
     *
     * @param magie Die Magie, die die Waffe verbraucht (bei dem
208  * Spielerwaffen)
     * @param schaden Den Schaden, den die Waffe anrichtet.
     * @param waffe Was für ein Typ ist die Waffe. Siehe das enum
211  * @param rof rate of fire - mit welchen Tick-abstand soll die
     * Waffe abgefeuert werden?
     * @param bez Wie die Waffe heisst. Nur für den Spieler
214  * @param id Der ID-Name der Waffe (der auch im Savegame abgespeichert
     * wird ggf.)
     * @param bild Unter welchem Namen ist die Darstellung dieser Waffe
217  * bekannt?
     * @param geschw Die Geschwindigkeit der Waffe
     * @param partikeltreff Abzuspielender Partikeleffekt beim treffen mit
220  * dieser Waffe
     * @param partikeltot Abzuspielender Partikeleffekt beim Töten mit
     * dieser Waffe
223  * @param startsound Welcher Sound soll beim starten dieser Waffe
     * gespielt werden?
     * @param hitsound Welcher Sound soll beim Aufschlag mit dieser Waffe
226  * gespielt werden?
     * @param totsound Welcher sound soll beim töten mit dieser Waffe
     * abgespielt werden?
229  * @param beben Wie viele Ticks soll der Bildschirm wackeln, wenn
     * ein Gegner mit dieser Waffe abgeschossen wurde?
     */
232 TWaffe(int magie, int schaden, Waffen waffe, int rof, String bez,
     String id, String bild, int geschw,
     TPartikelVerwaltung.Partikel partikeltreff,
235  TPartikelVerwaltung.Partikel partikeltot,
     String startsound, String hitsound,
     String totsound, int beben) {
238
     fmagie         = magie;
     fschaden       = schaden;
241  fwaffe         = waffe;
     frof           = rof;
     fbez           = bez;
244  fid            = id;
     fbild          = bild;
     fgeschw        = geschw;
247  fpartikeltreff = partikeltreff;
     fpartikeltot   = partikeltot;
     fstartsound    = startsound;
250  fhitsound      = hitsound;
     ftotsound      = totsound;
     fbeben         = beben;
253
     if ((bild != null) && (!bild.equals(""))) {
      TAnimation ani = TSharedObjects.getBild(bild);
```

```
256     if (ani != null) {
         try {
          ani.rotiereFrame();
259      } catch (Exception e) {
          System.out.println("Fehler in TWaffe: " + e);
         }
262     }
       }
      }
265
      /**
       * Erzeugt eine neue Waffe aus den Daten eines XML-Nodes
268    *
       * @param n Der Node, aus dem die Waffe erzeugt werden soll
       */
271   public static TWaffe WaffeAusNode(Node n) {
       int magie          = 0;
       int schaden        = -1;
274    int beben          = 0;
       Waffen waffe       = Waffen.NIL;
       int rof            = -1;
277    String bez         = "";
       String id          = "";
       String bild        = null;
280    int geschw         = -1;
       String startsound = null;
       String hitsound   = null;
283    String totsound   = null;
       TPartikelVerwaltung.Partikel partikeltreff =
        TPartikelVerwaltung.Partikel.NIL;
286    TPartikelVerwaltung.Partikel partikeltot =
        TPartikelVerwaltung.Partikel.NIL;

289    // in die Liste eintauchen
       NodeList nds = n.getChildNodes();
       Node a;
292
       for (int i = 0; i < nds.getLength(); i++) {
        a = nds.item(i);
295
        if (a.getNodeName().equals("magie")) {
         magie = Integer.parseInt(a.getTextContent());
298     } else if (a.getNodeName().equals("ID")) {
         id = a.getTextContent();
        } else if (a.getNodeName().equals("schaden")) {
301      schaden = Integer.parseInt(a.getTextContent());
        } else if (a.getNodeName().equals("beben")) {
         beben = Integer.parseInt(a.getTextContent());
304     } else if (a.getNodeName().equals("rof")) {
         rof = Integer.parseInt(a.getTextContent());
        } else if (a.getNodeName().equals("bezeichnung")) {
307      bez = a.getTextContent();
        } else if (a.getNodeName().equals("bild")) {
         bild = a.getTextContent();
310     } else if (a.getNodeName().equals("startsound")) {
         startsound = a.getTextContent();
        } else if (a.getNodeName().equals("hitsound")) {
313      hitsound = a.getTextContent();
        } else if (a.getNodeName().equals("totsound")) {
```

```
      totsound = a.getTextContent();
    } else if (a.getNodeName().equals("partikeltreff")) {
      partikeltreff = TPartikelVerwaltung.Partikel.valueOf(
      a.getTextContent());
    } else if (a.getNodeName().equals("partikeltot")) {
      partikeltot = TPartikelVerwaltung.Partikel.valueOf(
      a.getTextContent());
    } else if (a.getNodeName().equals("geschwindigkeit")) {
      geschw = Integer.parseInt(a.getTextContent());
    } else if (a.getNodeName().equals("waffe")) {
      waffe = Waffen.valueOf(a.getTextContent());
    }
   }

   if ((waffe == Waffen.NIL) || (schaden < 0) || (rof < 0) ||
    (bild == null) || (geschw < 0)) {
    System.out.println("Fehlende Angaben (" + waffe + ", " +
     schaden + ", " + rof + ", " + bild + ", " + geschw + ")");
    return null;
   }

   if ((bild != "") && (!TSharedObjects.bildInListe(bild))) {
    System.out.println("Bild der Waffe \"" +waffe + "\": \"" +
     bild + "\" nicht gefunden.");
    return null;
   }

   if ((startsound != null) && !TSound.inListe(startsound)) {
    System.out.println("Fehler in der Waffe \"" + bez + "\": Startsound \"" +
     startsound + "\" nicht gefunden.");
    // Hindert nicht Waffe an sich zu funktionieren, also weiter
    startsound = null;
   }

   if ((hitsound != null) && !TSound.inListe(hitsound)) {
    System.out.println("Fehler in der Waffe \"" + bez + "\": Hitsound \"" +
     hitsound + "\" nicht gefunden.");
    // Hindert nicht Waffe an sich zu funktionieren, also weiter
    hitsound = null;
   }

   if ((totsound != null) && !TSound.inListe(totsound)) {
    System.out.println("Fehler in der Waffe \"" + bez + "\": Totsound \"" +
     totsound + "\" nicht gefunden.");
    // Hindert nicht Waffe an sich zu funktionieren, also weiter
    totsound = null;
   }

   return new TWaffe(magie, schaden, waffe, rof, bez, id, bild, geschw,
    partikeltreff, partikeltot, startsound, hitsound, totsound, beben);
  }

  /**
   * Liefert den Magiewert, den die Waffe absaugt
   */
  public int getMagie() {
   return fmagie;
  }

```

```
    /**
     * Um welche Waffenart handelt es sich (aus Enum)
376  */
    public Waffen getWaffe() {
     return fwaffe;
379 }

    /**
382  * Liefert den Schaden, den die Waffe verursacht
     */
    public int getSchaden() {
385  return fschaden;
    }

388 /**
     * Liefert die Schussfrequenz der Waffe
     */
391 public int getRof() {
     return frof;
    }
394
    /**
     * Liefert die ID es Bildes
397  */
    public String getBild() {
     return fbild;
400 }

    /**
403  * Liefert die Fluggeschwindigkeit der Geschosse dieser Waffe
     */
    public int getGeschw() {
406  return fgeschw;
    }

409 /**
     * Liefert den Typ des Partikeleffekts, den ein Treffer eines
     * Geschosses dieser Waffe hervorrufen soll
412  */
    public TPartikelVerwaltung.Partikel getPartikelTreff() {
     return fpartikeltreff;
415 }

    /**
418  * Liefert den Typ des Partikeleffekts, den ein Abschuss eines
     * Gegners mit einem Geschoss dieser Waffe hervorrufen soll
     */
421 public TPartikelVerwaltung.Partikel getPartikelTot() {
     return fpartikeltot;
    }
424
    /**
     * Liefert den Name der Waffe, fuer HUD
427  */
    public String toString() {
     return fbez;
430 }

    /**
```

```
433     * Liefert die ID der Waffe, fuer Savegames
        */
       public String getID() {
436     return fid;
       }

439    /**
        * Liefert die ID des Sounds, der beim Abfeuern eines
        * Geschosses mit dieser Waffe gespielt werden soll
442     */
       public String getStartSound() {
        return fstartsound;
445    }

       /**
448     * Liefert die ID des Sounds, der beim Treffen eines
        * eines Gegners mit einem Geschoss dieser Waffe gespielt werden soll
        */
451    public String getHitSound() {
        return fhitsound;
       }
454
       /**
        * Liefert die ID des Sounds, der beim Abschiessen eines
457     * eines Gegners mit einem Geschoss dieser Waffe gespielt werden soll
        */
       public String getTotSound() {
460     return ftotsound;
       }

463    /**
        * Liefert die Anzahl der Ticks, die der Bildschirm wackeln soll,
        * wenn ein Gegner mit dieser Waffe abgeschossen wurde
466     */
       public int getBeben() {
        return fbeben;
469    }
}
```