Visualisierung der Eidolon Auswertung

VisEiA

Seminararbeit in Informatik
Universität Fribourg, Schweiz

Referent: Prof. Dr. Andreas Meier
Betreuer: Stefan Hüsemann

eingereicht von:
David Buchmann
rue de l'industrie 2
1700 Fribourg
david.buchmann@unifr.ch

David Buchmann

23. April 2002

Inhaltsverzeichnis

Kapitel 1

Aufgabenstellung

1.1 Zielsetzung und Output der Arbeit

Diese Arbeit untersucht die Verwendung einer Datenbank in einer eigenständigen Applikation. In einer ersten Phase werden unterschiedliche Systeme verglichen, in der zweiten soll ein vollständiges Programm geschrieben werden, mit dem am geeignetsten erscheinenden Ansatz aus Phase 1.

Das Programm der zweiten Phase soll Spieldaten für das Internet-Spiel Eidolon aufbereiten. Daher kommt der Name dieser Arbeit: VisEiA, **Vis**ualisierung der **Ei**dolon[1] **A**uswertung.

Das Programm soll folgende Funktionen haben:

- Wirtschaftsdaten einer vereinfachten Welt verarbeiten.
- Diese Informationen anschaulich darstellen.
- Geographische Bezüge der Daten visualisieren.
- Daten speichern, um Bezüge über den Zeitverlauf herzustellen.

Da dieses Programm anschliessend auch gebraucht werden soll, sind zudem folgende formale Anforderungen zu stellen:

- läuft stabil.
- intuitive Benutzung.
- einfache Installation, keine aufwendige Konfiguration.
- flexibel, um unterschiedliche Varianten des Spiels zu unterstützen.

[1] Eidolon wird in Kapitel 2 genauer erklärt.

1.2 Wissenschaftliche Fragen

Folgende Fragen sollen in dieser Arbeit untersucht werden:

Wie werden Informationen geographisch und zeitlich dargestellt?

Eidolon hat ein einfaches Wirtschaftsmodell. Trotzdem gibt es eine beachtliche Informationsflut für den Spieler.

Relevanz: An vielen Orten fallen Daten an, die sowohl einzeln - hier für eine Region - von Bedeutung sind, als auch als Werte für eine Statistik. Dem Benutzer muss ein guter Überblick geboten werden, er soll nicht in Detailinformationen ertrinken. Gleichzeit möchte er unkompliziert die relevanten Details erfahren können.

Beantwortung durch: Nach einer Analyse der darzustellenden Informationen wird in der zweiten Phase ein Interface entwickelt, das versucht, geeignete Darstellungsformen zu finden.

Welches System eignet sich für diese Aufgabe?

Das Programm soll gerade auch für Informatik - Unerfahrene geeignet sein. Für die Datenverwaltung ergibt sich daraus, dass sie ohne zusätzliche Datenbank - Installation funktionieren muss. Trotzdem sollte das nicht bedeuten, dass im Programm ein ganzes DBMS implementiert wird.

Relevanz: Diese Anforderung stellt sich bei den meisten Programmen für Heimanwender, die eine Datenbank verwenden. (Im Gegensatz zu Unternehmen, die auch komplexere Installationen in Kauf nehmen müssen.) Wichtig ist dabei Einfachheit für Anwender *und* Programmierer, bei möglichst grosser Zuverlässigkeit und einer befriedigenden Geschwindigkeit.

Beantwortung durch: Es gibt verschiedene Modelle, die hier in Frage kommen. In der ersten Phase wird recherchiert, welches hier geeignet ist. In der zweiten Phase wird ein Modell exemplarisch umgesetzt. Dabei wird sich zeigen, ob es den Kriterien wirklich gerecht wird.

Welche Programmierumgebung eignet sich für diese Aufgabe?

Relevanz: Alle aktuellen Programmiersprachen bieten eine Möglichkeit zur Anbindung an Datenbanken und haben graphische Komponenten. Die Qualität dieser Komponente ist eine wichtige Eigenschaft der Programmierumgebungen.

Beantwortung durch: Es werden die Ansätze der verschiedenen Sprachen und Umgebungen untersucht. Schwergewicht bildet der Zugriff auf Datenbanken, wodurch diese Frage mit der 2. Frage zusammenhängt. In der zweiten Phase wird für die Implementierung des Programms eine Umgebung gewählt, die sich dann in der Praxis zu bewähren hat.

1.3 Wer soll die Ergebnisse der Arbeit verwenden?

Das Programm wird den Spielern von Eidolon zur Verfügung gestellt. Wenn es gelingt, das Programm auch für andere Atlantis - verwandte Spiele zu konfigurieren, können es auch andere Spieler verwenden. Die Beantwortung der wissenschaftlichen Fragen ist für alle Informatiker von Interesse, die in ihren Programmen eine Datenbank verwenden möchten.

1.4 Zeitplan

31. 12. 2001	Datenbanksystem und Programmierumgebung ausgewählt Entsprechende Kapitel der Arbeit im Entwurf geschrieben
31. 1. 2001	Datenbankstruktur entwickelt
19. 3. 2001	Grundfunktionalitäten des Programmes implementiert
31. 4. 2001	Programm läuft stabil Rohentwurf der schriftlichen Arbeit
15. 7. 2001	Schriftliche Arbeit fertiggestellt Programm getestet und als Version 1.0 im Internet verfügbar

Kapitel 2

Atlantis und Eidolon als Internetkultur

2.1 Was ist Play by e-Mail?

Als eine Variante von Rollenspielen gibt es schon länger das Postspiel (Play by Mail, PBM), wo viele Spieler in einer Fantasiewelt agieren können. Diese Spiele werden, wie es der Name sagt, per Post gespielt: Die Spieler senden dem Spielleiter ihre Aktionen in einem Brief. Alle zwei Wochen oder noch seltener wertet dieser die Aktionen und Ereignisse aus und erstellt den neuen Zustand, den er an alle Spieler versendet.

Spiele, die genaue Regeln haben, können auch mit Computerprogrammen ausgewertet werden. Die automatische Verarbeitung erlaubt komplexe Regeln und viele Mitspieler. Als das Internet noch nicht weit verbreitet war, existierten Mischformen zwischen Papier- und elektronischen Spielen. [1]

Mit der starken Verbreitung von Internet in den USA und Westeuropa stiegen viele dieser Spielergemeinschaften auf den Computer und Email um. Der Hauptvorteil war der kleinere Zeitaufwand für den Organisator und geringere Kosten. Per Post musste für jeden Zug ein Briefporto bezahlt werden. Hat man die Infrastruktur des Internet sowieso, fallen höchstens noch minimale Telefonkosten zusätzlich an. Mit dieser Entwicklung hat sich die Zahl der Spiele und auch die Mitspielerzahl vergrössert.

Die Postspiele waren normalerweise kostenpflichtig. Wegen Briefmarken und Drucken von Auswertungen entstanden dem Organisator beträchtliche Unkosten. Dagegen sind viele Emailspiele gratis. Die Organisatoren machen ihre Arbeit als Hobby, es entstehen höchsten Kosten für einen Webserver, die durch Sponsoren oder Werbung gedeckt werden können. Diese Art Spiele wird analog zu den PBM als **PBeM** bezeichnet, **P**lay **b**y **e-m**ail, auf deutsch etwa Email-Spiel.

[1] Bei einem Spiel, das der Autor eine Zeit lang spielte, wurde der Zug mit einem Computerprogramm erstellt. Für den Versand an den Spielleiter wurde der Zug auf Diskette gespeichert und in einem Umschlag per Post gesendet. Die Auswertung erfolgte auf Papier, auf der Diskette waren die neuen Daten für das Zugerstell - Programm. Die Einführung von Email war für die nächste Version geplant.

2.2 Atlantis

Atlantis ist eines dieser PBeM Spiele. Es wurde von Russel Wallace entwickelt. Die Regeln und der Quellcode des Programmes stehen unter GNU General Public License [23] und sind frei zugänglich. Jedermann darf den Code verwenden, wenn sein Programm ebenfalls wieder unter der GPL steht. So hat Alexander Schröder das Spiel erweitert und auf Deutsch übersetzt. Seine Version heisst German Atlantis und ist unter [18] verfügbar. Mit diesem Programm hat Roman Meng für die Zeitschrift *Chronator* das Spiel Eidolon kreiert, das auf der Homepage der Zeitschrift[19] gratis angeboten wird.

2.2.1 Spielprinzip von Atlantis

Atlantis spielt in einer Fantasy - Welt. Jeder Spieler hat ein Volk, über das er befiehlt. Er rekrutiert aus den "Bauern" neue Untertanen, denen er seine Befehle gibt. Jede Gruppe (Einheit genannt) wird einzeln gesteuert. Sie kann verschiedene Fähigkeiten erlernen oder Aufgaben ausführen.

Die Welt besteht aus rechteckigen, schachbrettartig angeordneten Regionen. Einheiten können von Region zu Region wandern und Aktivitäten ausführen, z.b. Holzfällen, Steuern eintreiben, Burgen bauen oder neue Leute anwerben. Es gibt auch Meere, die nur mit Schiffen überquert werden können.

Die Einheiten können rund zwei Dutzend verschiedene Talente erlernen. Diese Fähigkeiten erlauben es, Güter herzustellen oder Geld zu verdienen, oder verbessern die Fortbewegung und die Chancen in einem Kampf. Es gibt verschiedene Rohstoffe: Holz, das durch das Fällen von Bäumen gewonnen wird sowie Steinquader und Eisenbarren, die in Bergregionen abgebaut werden. Daraus können mit den entsprechenden Fähigkeiten Güter wie Wagen, Burgen oder Schwerter hergestellt werden. Um die Kreativität zu fördern, kann der Spieler seine Figuren, Burgen, Schiffe und sogar die Regionen benennen und beschreiben. [2]

Es gibt in Atlantis kein vorgegebenes Spielziel. Jeder Spieler kann sich selber Ziele setzen. Da meist nur alle 2 Wochen ein Zug abgegeben wird, dauert ein Spiel über Jahre. Den Reiz des Spiels macht vor allem der Kontakt mit anderen Spielern aus, mit denen man gemeinsame Ziele zu erreichen versucht oder Handel treibt. Trotz den starren Regeln hat das Spiel eine rollenspielerische Komponente durch die menschlichen Mitspieler. In einer Welt sind beliebig viele Spieler möglich, es gibt Varianten von Atlantis, die über 500 Mitspieler haben, Eidolon hat immerhin rund 50.

2.2.2 Funktionsweise von Atlantis

Atlantis wird von einem Host verwaltet, einem Programm, das die ganzen Regeln enthält und die Daten des aktuellen Spieles verwaltet. Ein Spielleiter administriert den Host.

Die Spieler senden in einer Email Befehle für ihre Einheiten an den den Host. Dieser sammelt die Anweisungen von allen Spielern und verarbeitet sie am ZAT (Zugabgabe Termin). Der Host sendet jedem Spieler eine Auswertung, in der die Ergebnisse seiner

[2] Für eine ausführliche Anleitung sei auf die Webseiten von German Atlantis [18] und Eidolon [19] verwiesen.

Befehle und der neue Zustand beschrieben sind. Dieser Vorgang wird in Abbildung 2.1 veranschaulicht.

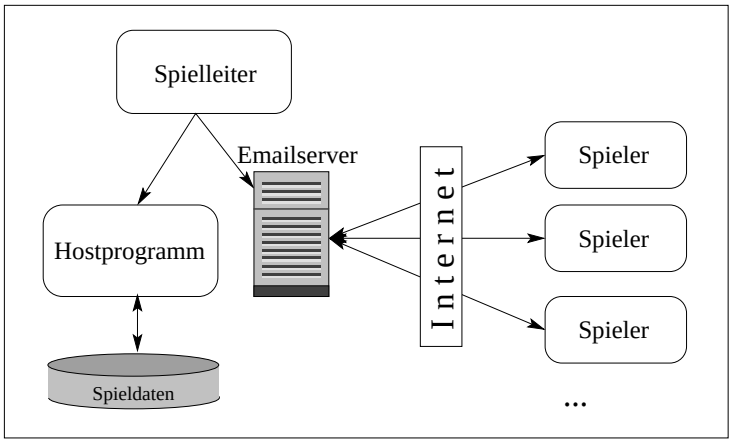


Abbildung 2.1: Play by Email Spiele werden über Internet gespielt und haben üblicherweise eine grosse Anzahl Teilnehmer

Das Spiel funktioniert auf Textbasis. Es gibt einige Tools, um diese Auswertung graphisch darzustellen, allerdings sind sie sehr beschränkt. Der Autor liess sich von zwei Programmen inspirieren: EidolonC[21], ein einfaches Programm um die Auswertung darzustellen, und AChart[22], das Statistiken mit mehreren Auswertungen macht.

Kapitel 3

Methoden der Datenspeicherung

Die meisten Programme müssen Daten konservieren, wenn sie beendet werden. Die einfachste Form, die auch heute noch Sinn machen kann, ist das Abspeichern in einer Datei. Für viele Daten existieren Standards, zum Beispiel wird die Kodierung von Buchstaben und Textsteuerzeichen mit den ASCII-Kodes standardisiert. Viele faktische Standards entstanden aus dem proprietären Format einer Firma, zum Beispiel die CompuServe GIF Bilddateien, die heute von fast allen Graphikprogrammen unterstützt werden.

Standards haben den Vorteil, dass kein eigenes Konzept entwickelt werden muss und dass häufig schon Anleitungen oder sogar fertige Programmelemente vorhanden sind, um mit den Daten zu arbeiten. Zudem wird die Bearbeitung der erzeugten Daten mit anderen Programmen möglich.

Das Konzept der Dateien funktioniert gut, wenn eine zusammenhängende Einheit, wie ein Text, ein Bild oder digitalisierte Musik, gespeichert werden muss. Es gibt aber Programme, die mit anderen Arten von Daten arbeiten. Ein Buchhaltungsprogramm hat lauter gleich aufgebaute Einträge; für Buchungen, Kontoangaben und so weiter. Die einzelnen Einträge sind kurz, eine Zahl, ein Wort oder vielleicht ein Satz. Die Daten haben eine tabellarische Struktur. Das Programm sollte die Daten strukturiert abspeichern, damit es noch weiss, was ein Datum ist und was ein Geldbetrag, wenn die Abrechnung erstellt werden soll. Wenn es um eine Unternehmensbuchhaltung geht, sollten zudem noch mehrere Anwender gleichzeitig auf die Daten zugreifen und sie ergänzen können, ohne dass Fehler entstehen. Natürlich kann man auch solche Daten direkt in eine Datei speichern. Man kann ein eigenes Dateiformat definieren und die entsprechenden Routinen schreiben. Allerdings läuft das darauf hinaus, dass man einen Grossteil der Funktionalität eines Datenbanksystems selber programmieren muss. Eine bessere Möglichkeit ist das Verwenden des XML Standards. Hier werden die Daten ebenfalls in einzelne Dateien gespeichert. Aber es ist ein sehr flexibles Format, das sorgfältig spezifiziert und standardisiert ist. Es gibt fertige Programmbausteine, um die Daten in diesem Format zu lesen, zu schreiben und zu verarbeiten.

Spätestens wenn mehrere Benutzer die Daten auch gleichzeitig verändern können müssen, reicht auch XML nicht mehr aus. Für diese Aufgaben wurden die Datenbanksysteme entwickelt. Diese eignen sich, um viele Benutzer gleichzeitig und ressourcenschonend arbeiten zu lassen.

In den folgenden Kapiteln werden diese Systeme erklärt und die konkreten Anwendungsmöglichkeiten besprochen.

3.1 XML (eXtensible Markup Language)

[1] XML ist eine vereinfachte Version von SGML (**S**tandard **G**eneralized **M**arkup **L**anguage). Der Standard wurde von den grossen Firmen der Computerbranche entwickelt, insbesondere für den Datenaustausch zwischen Programmen, gerade im Internet. Er erlaubt es, Informationen hierarchisch anzuordnen. In XML-Dateien wird diese Struktur mit Hilfe von Tags festgelegt. Das sind, wie bei HTML (**H**yper**t**ext **M**arkup **L**anguage), zwischen < und > eingeschlossene Schlüsselwörtern. Die erlaubten Tags werden in der DTD (**D**ocument **T**ype **D**efinition) definiert. Das bedeutet, man kann eine eigene Struktur von Tags definieren.

Für die Programmierer gibt es API's, die das DOM (**D**ocument **O**bject **M**odel) implementieren. Mit ihnen baut man eine hierarchische Datenstruktur auf, in der Tags, Attribute und Unterelemente durch Objekte dargestellt werden. Wegen dem Internetumfeld sind die meisten verfügbaren Programme in Java geschrieben.

Ein Beispiel für eine XML Datei:

```
<?xml version="1.0" standalone="yes"?>
<root>
   <node>
      Nummer 1
   </node>
   <node>
      Nummer 2
      <sub>
         BlaBla
      </sub>
   </node>
</root>
```

Das Einrücken ist nicht nötig, man könnte auch alles auf eine Zeile schreiben. Leerzeichen, Zeilenwechsel und ähnliches wird beim Einlesen ignoriert (sogenannter Whitespace). Wichtig ist, dass jeder Tag abgeschlossen wird.[2] Dazu wird der gleiche Name geschrieben, aber mit einem '/' vor dem Wort: </node>. Untergeordnete Elemente müssen geschlossen sein, bevor das übergeordnete geschlossen werden kann. Dies, weil die Daten hierarchisch sind. Eine Datei, die diese Forderung erfüllt, ist gut geformt (well-formed).

Im Unterschied zu HTML können die Elementnamen selber definiert werden. So kann man die eigenen Daten sehr genau strukturieren. Eine XML Datei kann für sich alleine stehen. Die erlaubte Struktur ist dann aus der vorhandenen Struktur abgeleitet. Normalerweise wird aber zuerst die DTD aufgestellt. Diese beschreibt, welche Elemente welchen untergeordnet sein dürfen oder müssen. Erfüllt eine XML Datei ihre DTD, so ist sie gültig (valid).

Die DTD für unser Beispiel könnte lauten:

```
<!ELEMENT root (node+)>
```

[1] Vgl. [13]

[2] Im Gegensatz zu HTML, wo ein Tag wie
 alleine stehen darf.

```
<!ELEMENT node (#PCDATA | sub)*>
<!ELEMENT sub (#PCDATA)>
```

Eine DTD zu definieren macht vor allem dann Sinn, wenn man die Daten automatisch verarbeiten will, und sich dazu auf die Dokumentstruktur verlassen muss.

Um XML darzustellen, gibt es verschiedene Möglichkeiten:

- CSS (Cascading Style Sheets) erlaubt die Beschreibung der Darstellung der Elemente in einem Webbrowser.
- XSL FO (Extended Stylesheet Language, Formating Objects) erlaubt ebenfalls, die Darstellung der Elemente zu beschreiben. Es gibt Programme, die aus einer FO Beschreibung und der XML Datei ein PDF für den Acrobat Reader erzeugen.
- XSLT (Extdended Stylesheet Language, Transform) übersetzt Elemente in andere. Damit kann man zum Beispiel eine HTML Datei erzeugen.

XSLT ist auch für andere Aufgaben als die Darstellung geeignet. Man kann einzelne Elemente aus einer XML Datei auswählen, sie anders sortieren etc. Für dieses Projekt interessant ist nur XSLT, da die Daten nicht in einem Webbrowser dargestellt werden.

Für das Verarbeiten von XML Daten mit eigenen Programmen gibt es zwei Standards, Simple API for XML (SAX) und Document Object Model (DOM). SAX wurde in einer Mailingliste der W3C definiert. Diese API eignet sich für ein serielles Einlesen der Daten aus einer XML Datei und das serielle Schreiben der Daten.

DOM wurde vom W3C selber definiert. Es beschreibt die Baumstruktur der hierarischen Daten der XML Datei. Dabei kann an jeder Stelle des Dokumentbaums ein Element oder ein Attribut eingefügt oder gelöscht werden. Das Dokument kann mit der grösstmöglichen Freiheit im Speicher bearbeitet werden.

Es existieren Bibliotheken für verschiedene Programmiersprachen, die einen Parser für SAX und die Klassen für DOM implementieren. Für Java gibt es die Referenzimplementation von Sun, das Project X [11]. Von der Apache Group gibt es die Open Source Projekte Xerces, ein validierender Parser und Xalan, ein XSLT Prozessor, auch in C und anderen Sprachen. Für das XML Umfeld bietet die XML Gruppe von Apache auch weitere interessante Projekte[12].

3.2 DBMS (Database Management Systems)

DBMS wurden entwickelt, um die immer wiederkehrenden Arbeiten mit Daten zu vereinfachen. Sie unterstützen beim zur Verfügung stellen, auswählen, sortieren und speichern von Daten. In einem DBMS sind die Daten zentral gespeichert und können so gut geschützt werden. (Sowohl vor Verlust als auch vor unbefugtem Zugriff.) Ein weiterer Vorteil ist, dass verschiedene Anwender mit unterschiedlichen Programmen gleichzeitig auf die Daten zugreifen können.

Sein volles Potential spielt ein Datenbanksystem erst aus, wenn es auf einem eigenen Computer läuft. Dieser Computer wird dann als Server bezeichnet, die Computer, die die Dienste des Datenbankservers verwenden, Clients. Für kleinere Aufgaben oder zum

Testen kann das Datenbanksystem auch auf dem selben Computer wie die Programme verwendet werden. In dieser Situation kann man das Datenbankprogramm als Server bezeichnen, das andere Programm als Client.

Wichtige Funktionen gehen aber verloren, wenn Client und Serverprogramme auf dem gleichen Computer laufen. Wenn der Server ein anderer Computer ist, kann die Datenbank in einem konsistenten Zustand gehalten werden, wenn ein Client abstürzt. Wenn alles auf dem gleichen Computer ist und dieser abstürzt, kann es sein, dass die Datenbank ungültige Daten enthält und im schlimmsten Fall unbrauchbar wird.

In diesem Projekt werden wir mit den Problemen von Datenbank und Client auf der selben Maschine leben müssen, da ein Heimanwender kaum einen extra Computer hat, um ihn als Datenbankserver zu verwenden.

3.3 Verwendung von Datenbanken in Programmen

Aktuelle Entwicklungsumgebungen sind mit Modulen zum Datenbankzugriff ausgestattet. Heutige Datenbanken werden meist mit SQL eingerichtet, abgefragt und verändert. Das Datenbanksystem kann eine Konsole anbieten, wo man die SQL-Befehle direkt eingeben kann. Normalerweise möchte man aber aus einer Programmiersprache heraus auf die Datenbank zugreifen. Um die Funktionalität der Datenbank zu nutzen, wird eine Bibliothek mit Funktionen zur Verfügung gestellt, die aus dem Anwendungsprogramm heraus aufgerufen werden können. Weil diese aber von System zu System verschieden sind, wurde eine Standardschnittstelle festgelegt. Dieser Standard heisst ODBC, er wird im nächsten Abschnitt genauer erklärt. Danach folgt die Vorstellung von JDBC, dem Konzept von Java. Zum Schluss wird eine Implementation der direkten Anbindung an eine Datenbank durch spezielle Bibliotheken an einem Beispiel untersucht.

3.3.1 ODBC, Open Database Connectivity

Die Abkürzung steht für Open Database Connectivity, auf Deutsch etwa offene Datenbank Verbindungsmöglichkeit. Der Standard wurde von Microsoft veröffentlicht, um zwischen den Datenbanken und den Anwendungsprogrammen eine wohldefinierte Schnittstelle zu etablieren. ODBC basiert auf einem offenen Standard der SQL Access Group (SAG).[3]

Für fast jedes DBMS gibt es einen ODBC Treiber, mit dem dann jede ODBC-fähige Anwendung zusammenarbeiten kann. ODBC wird zentral verwaltet. Es werden Aliase für einzelne Datenbanken vergeben, mit den Informationen über Pfad und Typ der Datenbank, zu verwendendem Treiber und weiteren Einstellungen. Die Anwendung kann sich mit einem Alias verbinden. Das System schaut dann nach, welcher Treiber verwendet werden muss. Ist sie einmal Verbunden, kann die Anwendung SQL Befehle absetzen. Das Konzept wird in Abbildung 3.1 veranschaulicht.

Der ODBC Treiber leitet das SQL an den Datenbankserver weiter oder übersetzt es in API Aufrufe für 'seine' Datenbank.

Für die Softwareentwickler ist dieser Standard auch nützlich, wenn eine Anwendung von einem Betriebssystem in ein anderes portiert werden muss. ODBC ist nicht auf die Windows-Welt beschränkt, sondern auch auf Unix, Macintosh und anderen Systemen

[3] Vgl. [24]

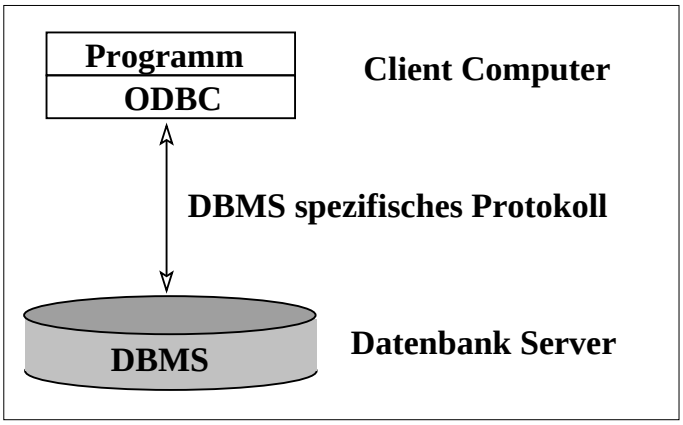


Abbildung 3.1: Das Konzept von ODBC

verfügbar. Auch wenn ein anderes DBMS installiert werden muss, zum Beispiel, weil die Datenbanken zu gross für das alte System werden, oder wenn ein spezieller Datenbankserver verwendet werden muss, bleibt der Zugriffsmechanismus auf ODBC derselbe.

Für ODBC stehen auf einem frisch installierten System keine Treiber zur Verfügung. Man kann einen Treiber extra installieren, manche Software wie Entwicklungsumgebungen oder MS Office enthalten ebenfalls welche. ODBC Treiber sind aber meist teuer, oft muss man für jedes System, auf dem sie installiert werden sollen, Lizenzgebühren bezahlen. Die freie Verteilung eines Programms wird damit fast verunmöglicht[4].

3.3.2 BDE, Borland Database Engine

Die BDE wurde von Borland für ihre Entwicklungsumgebungen geschrieben. Delphi und CBuilder enthalten Komponenten, um die BDE zu verwenden. Sie erfüllt die gleiche Aufgabe wie ODBC. Dazu enthält sie Treiber für den Datenbankzugriff und verwaltet Aliase. Sie kann auch ODBC Treiber verwenden.

Der grosse Vorteil ist der objektorientierte Ansatz der Komponenten. Während für ODBC Funktionen aufgerufen werden müssen, können mit der BDE einfach zu verwendende Objekte gebraucht werden. Wenn die BDE mit ihren eigenen Treibern arbeitet, ist sie sehr effizient, mit ODBC Treibern hingegen eher langsam.

Von Haus aus bringt die BDE Treiber für Access, Paradox, dBase und Textdateien mit, es gibt aber Treiber für andere Datenbanken zu kaufen.

3.3.3 IBX, Interbase Express

IBX ist eine Sammlung von Komponenten für Delphi/CBuilder, die direkt mit dem Interbase Server[5] arbeiten können. Sie verwenden weder BDE noch ODBC sondern rufen direkt Methoden aus der Interbase API auf. Sie sind gleich aufgebaut wie die BDE Klassen. Damit sind sie sehr intuitiv zu handhaben. Gleichzeitig ist der Datenzugriff ohne

[4]Es gibt ein Open Source Projekt für die Datenbank MySQL[16], das aber stark auf Linux ausgerichtet ist.

[5]Ein DBMS von Borland, das seit kurzem als Open Source zur Verfügung steht. Vgl. [14]

Umweg über die BDE schneller.

Nachteil ist, dass man auf Interbase beschränkt ist, und dass ein Interbase Server laufen muss. ODBC und BDE können auf die Standarddatenbanken (Paradox, dBase, Access) zugreifen, ohne dass ein Server laufen muss.

3.3.4 JDBC, Java Database Connectivity

[6] JDBC ist schon dem Namen nach nahe bei ODBC. Letzteres basiert aber auf C Bibliotheken und die Treiber sind jeweils für ein bestimmtes System kompiliert. Dies ist für Java nicht geeignet. Aufrufe in C Code führen zu Problemen mit der Sicherheit, der Robustheit und der Portabilität der Programme.

Sun hat deshalb für Java einen eigenen Standard geschaffen. Das Funktionsprinzip ist ähnlich wie bei ODBC. Für die Programmierung ist der grösste Unterschied, dass man mit Klassen arbeitet statt mit Datenstrukturen und Funktionen. Das Package java.sql definiert Interfaces, die ein Treiber implementieren muss, sowie Klassen und Interfaces für die Arbeit mit einer Datenbank. Der Treiber wird im Java-Code über das Interface angesprochen und kann zur Laufzeit ausgewählt und geladen werden.

Sun ist der Meinung, ihr System sei wesentlich benutzerfreundlicher als ODBC, das auch für einfache Aufgaben komplizierte Aufrufe verwende.

JDBC bietet zudem Unterstützung, um DBMS-Unabhängig zu arbeiten. Da SQL nicht von allen Herstellern gleich implementiert wurde, gibt es Abweichungen bei Datentypen, bei Bezeichnungen oder bei bestimmten Befehlen. JDBC enthält Mechanismen, um von diesen Details zu abstrahieren. Es ist aber auch möglich, die SQL Befehle direkt, also als nicht interpretierter String, zu übergeben.

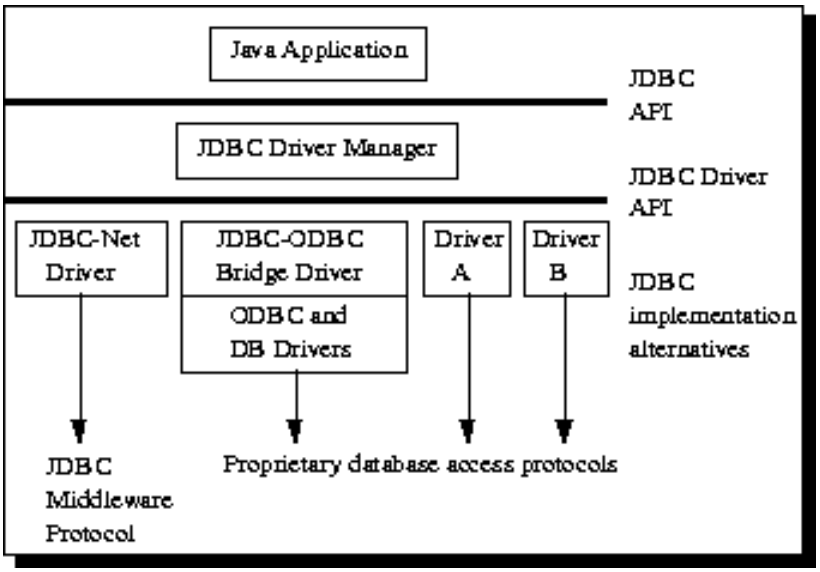


Abbildung 3.2: Die Vier Typen von JDBC Treibern. (Quelle: [10])

Es gibt vier Typen von JDBC Treibern (Siehe Abbildung 3.2):

[6] Vgl. [9] und [10]

1. *JDBC-ODBC bridge plus ODBC Driver.* Dieser Treiber stellt die Verbindung zu einem ODBC Treiber her. Damit kann man in Java bequem mit dem java.sql API arbeiten und einen verfügbaren ODBC Treiber verwenden. Der ODBC Treiber ist aber nicht in Java geschrieben und muss separat installiert und konfiguriert werden. Eine Implementation von Sun dieser "Brücke" ist in der Java Distribution enthalten.

2. *Native API partly-Java driver.* Dieser Treiber ersetzt den ODBC Treiber und macht selber Aufrufe in die API einer Datenbank. Die Probleme mit Nicht-Java Code bleiben bestehen, da bloss dem Systemtreiber eine Java Schicht aufgesetzt wurde.

3. *JDBC-Net pure Java driver.* Ein solcher Treiber verwendet das Herstellerunabhängige net Protokoll, das von einem Server übersetzt und an ein DBMS weitergeleitet wird. Damit hat man ein sehr flexibles System, bei dem Clients, zum Beispiel als Applet, auf jedem Java System laufen.

4. *Native-protocol pure Java driver.* Die letzte Lösung ist ein Treiber, der direkt mit dem DBMS eines bestimmten Hersteller aufnimmt und dessen Protokoll verwendet. Der Treiber ist vollständig in Java geschrieben und portabel. Er funktioniert aber nur mit einem bestimmten DBMS.

3.4 Programmierumgebungen

Für eine Anwendung wie VisEiA muss die Umgebung den Zugriff auf Daten sinnvoll unterstützen und eine gute graphische Darstellung erlauben. Untersucht werden einige Umgebungen, die zur Zeit des Projekts für den Autor verfügbar waren und sich eventuell eignen können.

3.4.1 Pascal (Delphi)

Delphi ist eine graphische Programmierumgebung für Windows von Borland / Inprise[6]. Als Programmiersprache wird der Pascaldialekt Object Pascal verwendet. Wichtigstes Element für die bequeme Programmierung ist die VCL (Visual Component Library), die vordefinierte Fenster- und Datenbankobjekte enthält. Diese erlaubt das sogenannte RAD (Rapid Application Developement). Man kann Fenster nicht nur im Code erzeugen, sondern auch mit der Maus zeichnen und Schaltflächen einfügen. Das Entwickeln graphischer Interfaces verbraucht normalerweise sehr viel Zeit, durch RAD kann dieser Schritt beschleunigt werden.

Die gleiche VCL ist auch im CBuilder, der C/C++ Entwicklungsumgebung von Borland verfügbar. Insofern sollten Delphi und CBuilder für diese Aufgaben gleichwertig sein. Delphi wird beiseite gelassen, weil der Autor C++ besser kennt als Pascal.

3.4.2 C++ (CBuilder)

C++ ist eine objektorientierte Programmiersprache, die auf C aufbaut. Es ist eine sehr mächtige Sprache, die mit hohem Abstraktionsniveau verwendet werden kann, aber auch für sehr systemnahes Programmieren. Die Speicherverwaltung muss teilweise vom Programmierer selber gemacht werden, was zwar hohe Effizienz der erzeugten Programme

erlaubt, aber ein grösseres Fehlerrisiko mit sich bringt. Es ist zum Beispiel möglich, versehentlich auf falsche Speicheradressen zuzugreifen.

CBuilder ist eine graphische Programmierumgebung von Borland/Inprise[6], analog zu Delphi. Der C/C++ Compiler unterstützt ANSI C / ANSI C++ . Daneben kennt er einige Erweiterungen von Borland. Diese Erweiterungen wurden eingeführt, um den Code zu optimieren und Konstrukte wie ein `try .. finally` zu erlauben. Andere Erweiterungen wurden gemacht, um die VCL von Delphi, die in Pascal geschrieben ist, verwenden zu können. Der Compiler kann also auch mit Pascal Bibliotheken umgehen, was aber für dieses Projekt nicht von Bedeutung ist.

3.4.3 Java

Java ist eine einfache objektorientierte Programmiersprache, die von Sun Corporation[8] definiert wurde. Wichtigste Eigenschaften sind die Möglichkeit, die kompilierten Programme auf allen System laufen zu lassen und die rigorose Standardisierung mit einem durchdachten Sicherheitskonzept. Java Programme werden nicht direkt vom Prozessor ausgeführt, sondern laufen auf einer sogenannten Java Virtual Machine. Die JVM ist ein Programm, dass den Java Code interpretiert. Java erlaubt keinen expliziten Zugriff auf den Speicher, wodurch viele Fehlerquellen ausgeschlossen werden. Die JVM verwaltet den Speicher mit Hilfe eines Garbage Collectors, eine Routine, die periodisch nicht mehr benötigten Speicher freigibt. Programmen, denen man nicht trauen kann, zum Beispiel in einem Browser aus dem Internet geladene, werden einzelne Operationen wie der Zugriff auf das Dateisystem verwehrt.

Der Nachteil ist, dass ein Java Programm nur mit Hilfe der JVM ausgeführt werden kann, diese also auf dem System installiert sein muss. Java Programme sind wegen dem Zusatzaufwand für Interpretieren und Garbage Collection langsamer, wobei das mit zunehmend schnelleren Computern an Bedeutung verliert.

Kapitel 4

Auswahl des geeigneten Systems

4.1 Verfahren

Aus diesen verschiedenen Systemen eine gute Kombination auszuwählen, ist schwierig. Jedes hat offensichtliche Vor- und Nachteile und kann verborgene Tücken aufweisen. Um die Funktionsweise der Systeme besser kennen zu lernen, werden einige Kombinationen getestet. Dies mit einem einfachen Beispiel, das keinen grossen Programmieraufwand erfordert.

Folgende Daten sollen verarbeitet werden: Wir haben Regionen, die einen Namen und eine Nummer haben. Weiter haben wir Einheiten, die ebenfalls eine Nummer haben und einen Namen. Die Einheiten sind eindeutig einer Region zugeordnet. (Wie wir später sehen werden, ein vereinfachtes Modell der Aufgabe, die VisEiA lösen muss.) Das Programm soll die Daten künstlich erzeugen, abspeichern und dann wieder laden. Neben der Performance interessiert vor allem, wie kompliziert der benötigte Code zu schreiben ist.

Diese Struktur soll nun mit einigen Systemen verarbeitet werden:

- Paradox Datenbank, via Paradox Treiber der BDE aus CBuilder.
- Access Datenbank, via ODBC Treiber durch die BDE aus CBuilder.
- Interbase Datenbank, direkt via IBX aus CBuilder.
- Advantage Database Server, direkt via Adv Komponenten
- XML - Datei, mit Java unter Verwendung des Sun XML-Parser (Project X).
- Access Datenbank, via Sun JDBC-ODBC Bridge und Microsoft ODBC Treiber aus Java.

4.1.1 Daten

Die Datenbank enthält zwei Tabellen:

```
Region  (RNummer SMALLINT, RName CHAR(10))
Einheit (ENummer SMALLINT, EName CHAR(10), RNummer SMALLINT)
```

Die DTD für die XML Datei ist

```
<!ELEMENT Region (Einheit*)>
<!ATTLIST Region
          RNummer CDATA,
          RName CDATA>
<!ELEMENT Einheit (EMPTY)>
<!ATTLIST Einheit
          ENummer CDATA,
          EName CDATA>
```

Die Regionen bekommen Nummern ab 1, die nächste immer +3, damit nicht jede Nummer exisitert. Der Name ist "Name " und dann die Nummer. Pro Region soll es 3 Einheiten geben, die Nummern ab 2 mit Schritten von +7 erhalten. Der Name ist ebenfalls "Name " und die Einheitsnummer.

Auf die Daten sollen folgende Operationen ausgeführt werden:

- Verbinden
  - Datenquelle auswählen.
  - Tabellen Region und Einheit werden gelöscht.
  - Tabellen Region und Einheit werden neu definiert.
- Trennen: Von der Datenquelle trennen.
- Die Daten mit einem vorbereiteten Statement einfügen.
- Eine Liste der Regionsnamen und der sich darin befindenden Einheiten für Regionsnummern <= 50 erstellen.
- Die Summe der Regionsnummern aller Einheiten mit gerader Nummer kleiner 147 berechnen.
- Die Tabelleninhalte löschen.

4.1.2 Programme

Die Programme werden möglichst ähnlich geschrieben, damit die Unterschiede in der Funktionalität sind und nicht im Design. Die beiden Java bzw. die beiden CBuilder Programme haben das gleiche Gerüst, aber auch zwischen den Java und den CBuilder Programmen besteht kein grosser struktureller Unterschied.

Um die Programme zu vergleichen, sollte auch der beiliegende Sourcecode angeschaut werden.

Java: jdbcTest

Um keinen JDBC Treiber und einen ganzen Datenbankserver zu benötigen, soll die Sun JDBC-ODBC Brücke verwendet werden. Sie ist Teil der normalen Java Distribution, muss also nicht separat gesucht werden. Sie ist in der Klasse sun.jdbc.odbc.JdbcOdbcDriver

definiert. Als ODBC Treiber wird die Microsoft Jet Engine verwendet, die mit MS Office installiert wird und ohne separaten Server auf Access Datenbanken zugreifen kann.

Das graphische Interface wird mit javax.swing Klassen erstellt.

Java: xmlTest

Für xmlTest wird ein XML Parser benötigt. Hier wird JAXP verwendet, die Referenzimplementation für SAX/DOM von Sun [8]. Etwas umständlich ist das Einbinden von Bibliotheken, die nicht standardmässig in Java enthalten sind. Die Pfade müssen an der Kommandozeile beim Aufruf angegeben werden. Um das Bequemer zu machen, wird ein Makefile erstellt. Mit dem Programm make, das unter Unix standardmässig installiert ist, wird das Programm aufgerufen. `make all` kompiliert das Programm und startet es anschliessend.

Es wird die gleiche Oberfläche verwendet wie jdbcTest und dieselben Funktionsnamen.

CBuilder: BDETest

Mit Hilfe der VCL Datenbankklassen wird die Anbindung an die BDE geschaffen. Damit kann jede Datenbank geöffnet werden, die als BDE oder ODBC Alias dem System bekannt ist.

Für das Interface werden ebenfalls Klassen der VCL verwendet.

CBuilder: IBXTest

Mit den Klassen aus dem Interbase Express Paket wird ein Programm sehr ähnlich dem BDETest geschrieben. Diese Klassen verwenden nicht die BDE, sondern arbeiten über ein proprietäres Verfahren mit dem Interbase Server zusammen. Offensichtlicher Nachteil ist, dass dieser Server separat gestartet werden muss.

CBuilder: AdvTest

Eine zusätzliche Möglichkeit bot sich mit der Beta Version eines Herstellers Namens Extended Systems[15]. Ihren 'Advantage Local Server' wurde beworben als Möglichkeit, einfach zu verteilende Datenbankanwendungen zu schreiben. Wenn man mit dem Programm einige Dateien mitgiebt, würde das Programm laufen, auch ohne die etwa 5 MB grosse Bibliothek für BDE. Eine Gruppe von Objekten analog zu denen von CBuilder würde den Zugriff auf eine Datenbank erlauben. Die Version, die ohne separaten Server arbeitet, kann gratis heruntergeladen werden. Die Hoffnung von Extended Systems ist, dass man für eine grössere Anwendung dann ihren richtigen Datenbankserver verwendet, der natürlich nicht mehr gratis ist.

Die Installation in die Entwicklungsumgebung läuft automatisch und sauber. Die Komponenten sind sehr ähnlich wie die aus BDETest und IBXTest. Leider ist der verwendete SQL - Dialekt (StreamlineSQL) stark eingeschränkt. Er kennt keinen `PRIMARY KEY`, kein `SMALLINT` etc.

4.2 Testergebnisse

4.2.1 DBMS basierte Programme

Die Projekte BDE/ODBC Datenquellen mit CBuilder und ODBC Quellen mit Java anzusteuern, sind geglückt. Die beiden Programme sehen sich recht ähnlich und liessen sich mit vertretbarem Aufwand erstellen. Die ausführbare Datei für das Programm BDE-Test ist 59 Kilobyte gross, die Java .class Dateien von jdbcTest belegen zusammen mit den Icons 30 Kilobyte. Natürlich laufen beide Programme nicht eigenständig. jdbcTest braucht die Standardbibliotheken von Java, die Java Foundation Classes und den ODBC Treiber, BDETest die Klassenbibliotheken von CBuilder und der BDE.

Wie man an den Abbildungen auf Seite 22 sieht, sind die optischen Unterschiede sehr gering, sogar zwischen den CBuilder- und Java-Programmen. Die Speicherung der Daten und ihre graphische Darstellung sind weitgehend unabhängig. (Anders sieht es aus, wenn man vorhandene Bibliotheken zur Datenrepräsentation verwenden möchte, die z.B. direkt mit XML oder einer Datenbank arbeiten.)

jdbcTest: Zugriffe auf ODBC Datenquellen mit JDBC

Der Treiber wird mit `Class.forName()` in den `DriverManager` geladen. Die Verbindung wird danach über den `DriverManager` geöffnet. Für Abfragen werden Klassen aus der Verbindungsinstanz von `Connection` erzeugt. Damit ist automatisch klar, auf welche Datenbank sie sich beziehen.

- Access (odbcjt32.dll): Funktioniert gut, allerdings wird die `PRIMARY KEY` Anweisung ignoriert.
- Excel (odbcjt32.dll): Der Treiber unterstützt kein `CREATE TABLE`.
- dBase (odbcjt32.dll): Statt Ganzzahlwerten werden Gleitkommazahlen retourniert.
- Paradox (odbcjt32.dll): Die Anweisung `PRIMARY KEY` wird nicht erkannt und als Fehler bezeichnet.

Wie für jedes Java Programm muss zum Ausführen das Java Runtime Environment oder das SDK installiert sein. Dann muss im ODBC-System eine Datenquelle konfiguriert sein. Der getestete Treiber 'Microsoft Jet Base' (odbcjt32.dll) kann Access, Excel, dBase, Paradox und Textdateien als Datenquellen verwenden. Allerdings funktioniert er nicht mit allen besonders gut.

BDETest: Zugriffe auf BDE Datenquellen aus CBuilder

CBuilder enthält Installshield Express, mit dem eine automatische Installation für mit CBuilder erstellte Programme konfiguriert werden kann. Die benötigten Bibliotheken für CBuilder und für den Datenzugriff mit BDE kann man hier bequem auswählen. Die Installation wurde ebenfalls erfolgreich getestet.

Die BDE bringt einen Standardtreiber für dBase, Paradox, Foxpro und Textdateien und einen für Access. Es zeigt sich aber, dass auch hier kleine Details in der SQL Syntax die Kompatibilität beschränken.

- Paradox (STANDARD): Funktioniert gut
- FoxPro (STANDARD): `CREATE TABLE` funktioniert nicht
- dBase (STANDARD): Problem mit dem Schreiben der Daten (Datentypenproblem)
- Access (MSACCESS): Die Syntax der Felddefinitionen in `CREATE TABLE` ist nicht Standard SQL.

Der Zugriff von der BDE auf ODBC funktioniert von den getesteten Systemen nur für Access gut, allerdings auch dort extrem langsam.

- Access (BDE, odbcjt32.dll): Funktioniert teilweise, aber sehr langsam. Die CAST Anweisung wird nicht unterstützt.
- Excel (BDE, odbcjt32.dll): Der Treiber unterstützt kein `CREATE TABLE`.
- Paradox (BDE, odbcjt32.dll): Das Programm hängt und muss per Taskmanager abgebrochen werden.
- dBase (BDE, odbcjt32.dll): Das Programm hängt und muss per Taskmanager abgegebrochen werden.

IBXTest: Verwenden der API von Interbase

Diese Komponenten greifen nicht direkt auf die Datenbank zu, sondern brauchen einen laufenden InterBase Server. Dieser ist zwar seit Version 6.0 ein Open Source Projekt geworden und frei verteilbar, doch ist die Lösung zu aufwendig. Die Installation würde problematisch. Bei den IBX Komponenten ist eigentlich eine Installationskomponente vorgesehen, mit der man angepasste Installationen durchführen könnte. Doch diese wurde leider noch nicht implementiert. Das Programmpaket würde durch Interbase auch unnötig vergrössert. Grösstes Problem wäre die Anwendung des Programmes. Damit es sich mit der Datenbank verbinden kann, muss zuvor der Interbase Datenbankserver gestartet werden.

Bei der Arbeit mit den Komponenten viel auf, dass die Fehlermeldungen ungenau und wenig hilfreich sind.

AdvTest: Advantage Database

Die Lösung mit dem Extended Systems Inc. Advantage Local Server wäre nicht schlecht. Allerdings wirkt das System nicht sehr zuverlässig. Die Komponenten für CBuilder sind fehlerhaft. So ist es nicht möglich, wenn man das Programm neu gestartet hat, ein INSERT Statement auszuführen (Advantage Server file open failure). Wenn zuerst ein `SELECT` gemacht wird, passiert der Fehler nicht mehr. Noch eigenartiger ist, dass dieser Fehler nicht jedes Mal auftritt. Ebenfalls erstaunlich ist, dass nach einer misslungenen DROP TABLE Anweisung die nachfolgenden `CREATE TABLE` Anweisungen zwar ausgeführt wurden, `INSERT` aber nicht durchgeführt wurde (ohne Fehlermeldungen). Wurde das Programm beendet und wieder gestartet, so funktionierte es wieder korrekt.

Unterschiede in der SQL Syntax

Der odbcjt32.dll Treiber von Microsoft unterstützt die `CAST` Anweisung nicht. Dafür versteht er einen Vergleich wie `RNummer LIKE '%2'`. RNummer ist ein `SMALLINT`. Der BDE Treiber wendet `LIKE` nicht auf Integertypen an, dafür funktioniert die Anweisung `CAST(RNummer AS CHAR(10)) LIKE '%2'`.

Der BDE Treiber akzeptiert bei `DELETE` kein *. Um alles zu löschen, braucht man `DELETE FROM Einheit`. Diese Syntax wird aber wiederum vom Microsoft Treiber für Paradox Tabellen nicht verstanden, dort braucht es den *.

4.2.2 XML basiertes Programm: xmlTest

Weiter wurde ein Java Programm für einen XML Parser geschrieben. Das Programm funktioniert ebenfalls zufriedenstellend. Weil hier direkt mit der Baumstruktur des DOM gearbeitet wurde, war der Programmieraufwand wesentlich grösser. Das Erstellen der Daten ist viel aufwendiger, erst recht die Abfragen, die gänzlich selber als Operationen auf die Baumstruktur implementiert wurden. (Wie dem Autor unterdessen bekannt wurde, wäre für Abfragen die Verwendung von XSL angebracht. Doch mit dem Wissensstand von Anfang Wintersemester wäre das Projekt mit XML nicht gut geglückt.)

4.3 Leistungsmessung

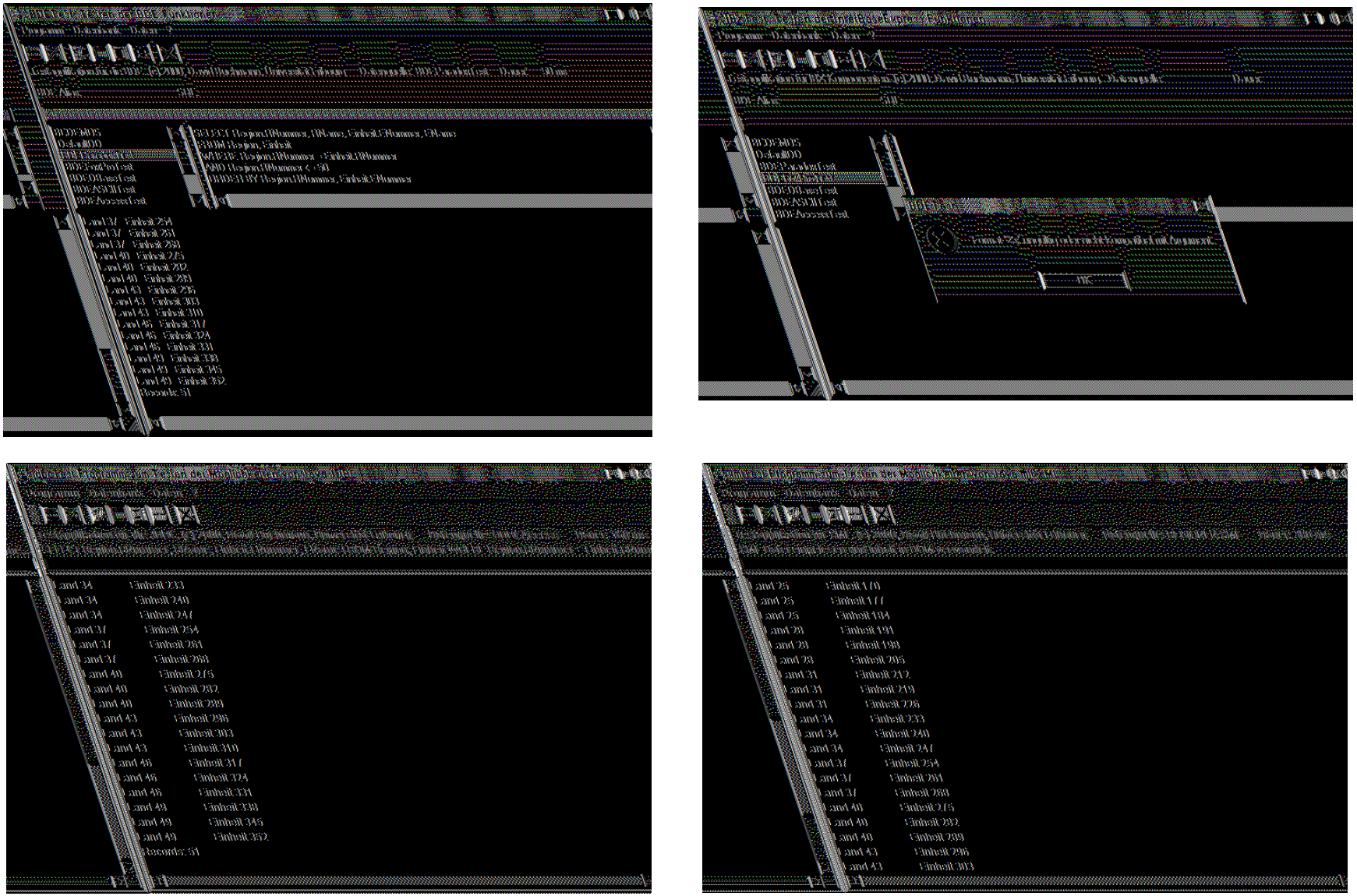
Testsystem: Windows NT 4.0, SP6, Athlon 500 MHz, 128 MB RAM

	JDBC-ODBC	BDE	BDE-ODBC	Advantage	XML
	MS Jet Engine Access Datei	Standard Paradox	MS Jet Engine Access Datei	Extended Sys Local Server	DOM Sun Proj. X
Verbinden	90 ms	10 ms	5170 ms	0 ms	200 ms
Schreiben	1400 ms	190 ms	4010 ms	1160 ms	150 ms
Liste lesen	40 ms	200 ms	50 ms	50 ms	200 ms
Summe	10 ms	10 ms	-fehler-	20 ms	60 ms
Löschen	0 ms	10 ms	0 ms	440 ms	10 ms
Trennen	10 ms	0 ms	150 ms	0 ms	0 ms

Diese Werte sind allerdings ungenau. Windows und die BDE arbeiten mit Puffern, die wiederhohlte Lesevorgänge bei den Datei- respektive den Datenbankzugriffen beschleunigen. Da das Betriebssystem im Hintergrund ab und zu Prozesse ausführt, kann auch die Systemleistung kurzfristig einbrechen. So kann ein Wert durchaus zwischen 50 und 250 ms schwanken. Die Werte sind also nicht unbedingt reproduzierbar, für eine grobe Einschätzung reichen sie.

Der Zugriff über BDE auf ODBC (Access) musste unter Windows 98 getestet werden. Die ODBC Treiber unter Windows NT 4.0, SP6, melden einen allgemeinen Syntaxfehler in der Tabellendefinition. (Die Version unter NT unterstützt `PRIMARY KEY` und `FOREIGN KEY` nicht. Erstaunlicherweise funktioniert unter Windows 98 `PRIMARY KEY` nur über die BDE, nicht aber über JDBC.) Das Summieren funktioniert nicht, da für die Selektion

Abbildung 4.1: Die vier Testprogramme sehen sich relativ ähnlich. Die Logik dahinter ist aber unterschiedlich.



ein Vergleich mit Hilfe von CAST verwendet wird. Wie bereits erwähnt, unterstützt die Microsoft Jet Engine diesen Operator nicht. Dafür unterstützt der BDE Treiber STANDARD die Optionen `NOT NULL` und `FOREIGN KEY` nicht.

`PRIMARY / FOREIGN KEY` werden ebenfalls nicht unterstützt, `PRIMARY KEY` lässt sich ersetzen durch das Erstellen eines `UNIQUE INDEX`, der das Einfügen von doppelten Schlüsseln ebenfalls verhindert.

4.4 Folgerungen

Was sehr schnell aufgefallen ist, sind die Probleme mit ODBC. Im Prinzip ist die Syntax immer ähnlich, doch 'ähnlich' reicht nicht. Wenn SQL im Programm verwendet werden soll, ist der kleinste Fehler ein Problem, das nicht automatisch gelöst werden kann. Das Programm, das mit allen Datenbanken gleich funktioniert, scheint es nicht zu geben. Für spezifische Treiber könnten spezielle SQL Statements geschrieben werden, was aber das grundlegende Problem nicht löst. Falls man mit beliebigen ODBC Datenquellen arbeiten möchte, wird es schwierig.

Die Zeitmessungen geben keine eindeutige Entscheidung vor, ausser den schlechten Werten des ODBC Zugriffs der BDE scheinen alle Varianten machbar. Die Lösung mit XML zeigte allerdings, dass der Datenzugriff dort auf wesentlich tieferem Niveau erfolgt. Man muss sich um das Abspeichern selber kümmern. Ein Teil der Einfachheit von JDBC bzw. BDE kommt von den bequemen Klassen, die die meiste Arbeit abnehmen. Einiges davon könnte wohl relativ einfach programmiert werden. Doch die komplizierteren SQL Abfragen wären sehr aufwendig, wenn das Durchsuchen des DOM - Baumes immer wieder von neuem geschrieben würde. Angesichts der strikt hierarchischen Struktur von XML sollten die Daten zudem auf meherere Dateien verteilt werden. Das alles scheint recht aufwendig im Vergleich zu den Datenbank - Lösungen. XML käme allenfalls für den Austausch zwischen Spielserver und Client in Frage, doch wird in diesem Projekt nur der Client geschrieben.

Advantage Local Server bietet nicht den versprochenen Vorteil. Ob der Server schlecht ist oder nur die angebotenen CBuilder Komponenten, kann nicht festgestellt werden. So oder so ist das Packet aber viel zu unzuverlässig. Somit bleiben JDBC und BDE als Möglichkeiten. Sie sollen nochmals gegenüber gestellt werden:

	JDBC	BDE
SQL	Mit Access kein `PRIMARY / FOREIGN KEY`, kein `CAST`	Paradox unterstützt kein [NOT] NULL
Geschwindigkeit	Gut, schreiben etwas langsam	Gut, lesen überdurchschnittlich langsam
Installation	ODBC Treiber muss installiert sein. ODBC Alias kann nicht direkt erzeugt werden.	Installshield Express kann verwendet werden. BDE und Treiber sowie Alias können dabei direkt installiert werden.
Programmierung	Programmierung der graphische Oberfläche "von Hand". Klassen ableiten und instanzieren. (Graphische Editoren vorhanden, können aber keine portablen Anwendungen erzeugen.)	Graphisch im CBuilder. Komponenten (spezielle Klassen) einstellen und instanzieren.

Das grosse Problem von JDBC ist, dass keine Treiber existieren, die ohne separaten Datenbankserver arbeiten. Zudem gibt es nur wenige frei verfügbare Treiber, die meisten würden eine Lizenzgebühr verlangen. Will man die JDBC-ODBC Brücke verwenden, so muss das ODBC System funktionieren.

ODBC Treiber wie die Microsoft Jet Engine dürfen nicht frei verteilt werden, es muss eine Lizenz gekauft werden. Die Jet Engine würde den potentiellen Nutzerkreis faktisch auf Benutzer von MS Office einschränken.

Dagegen bietet CBuilder eine sehr gute graphische Umgebung und mit der BDE ein System zum Datenbankzugriff, das mit selber geschriebenen Applikationen frei verteilt werden darf. Zudem ist im Programmpaket von CBuilder das Programm InstallShield Express enthalten, das eine profesionelle Installationsroutine (inklusive Deinstallation) anbietet.

Unter Anbetracht der Vor- und Nachteile soll das Programm in C++ mit dem CBuilder und der BDE realisiert werden. Der STANDARD Treiber der BDE soll verwendet werden, um Tabellen im Paradox Format zu erstellen, ohne dass ein separates Datenbankprogramm laufen muss.

Kapitel 5

Konzept

Unabhängig von der verwendeten Programmiersprache muss die Struktur der Daten, mit denen VisEiA arbeiten soll, verstanden werden. Diese soll mit Hilfe des Entitäten-Beziehungsmodells beschrieben werden.

Die Datenstruktur wird dann auf die Datenbank abgebildet und in C++ Klassen übersetzt.

Anschliessend wird die Gliederung des Programmes in funktionell unabhängige Schichten entworfen.

5.1 Datenmodellierung

Da die Verwendung einer relationalen Datenbank vorgesehen ist, werden die Daten nach dem Entitäten-Beziehungsmodell[1] gegliedert. Dieses erlaubt anschliessend eine einfache Überführung in ein relationales Datenbankschema.

Die Bezeichnungen für Entitäten, Attribute und Beziehungen werden in Englisch vergeben, um im Programm durchgehend englische Namen verwenden zu können. Es kann sich lohnen, zuerst das fertige ER-Modell in Abbildung 5.1, Seite 30, anzuschauen, um die folgenden Ausführungen besser zu verstehen.

5.1.1 Entitäten

Runde (Round)

Atlantis ist rundenbasiert, deshalb liegt es nahe, dass die Runde (*Round*) eine eigene Entität ist. Eine Runde dauert im Spiel einen Monat. Das Spiel beginnt im Januar des Jahres 1, man kann also die Runden auch von 1 an durchnummerieren. Da in der Auswertung sowieso die Rundennummer angegeben ist, wird diese verwendet (*RoundNr*). Die Information, welche Nummer die eigene Partei hat (*OwnFaction*), ist sehr nützlich für eine angenehme Benutzerführung.

Weiter erhält man in der Auswertung das Datum für die nächste Zugabgabe (*ZAT*, **Z**ug **A**bgabe **T**ermin) und ein Passwort (*Password*), das in diesem Zug angegeben werden muss.

[1] Eine gute Einführung in das Entitäten-Beziehnungsmodell gibt [4]

Für VisEiA soll zudem noch das Datum, wann die Auswertung geladen (*loaded*) wurde, und ein Kommentar (*Comment*) des Anwenders gespeichert werden.

Regionen (Region)

Eine Region hat Koordinaten in einem karthesischen System, also die Werte X und Y[2]. Weiter hat sie einen Namen (*Name*) und eine Beschreibung (*Description*). Zur Verarbeitung kann es auch interessant sein, zu speichern, wann die Region das letzte Mal besucht wurde (*LastVis* für last visited).

In einer Region gibt es Ressourcen wie Bäume oder Pferde und wirtschaftliche Angaben wie das maximal erzielbare Einkommen. Nicht nur die Ressourcen, sondern auch die Wirtschaftszahlen werden als Gegenstände betrachtet. Das hält die Region einfach und erlaubt zudem eine unkomplizierte Erzeugung der Statistiken.

Einheit (Unit)

Als weitere wichtige Entität drängt sich die Einheit (*Unit*) auf. Jede Einheit hat eine eindeutige Nummer, die sich folglich als Primärschlüssel eignet. Weitere Attribute sind *Name* und Beschreibung (*Description*). Zudem hat sie zwei Verhaltensregeln: sie kann eine Region bewachen (Boolean *guard*) und sie kann bei Schlachten vorne mitkämpfen, im Hintergrund bleiben oder versuchen, sich ganz aus dem Kampf herauszuhalten (Aufzählung *fights*).

Eine Einheit umfasst eine variable Anzahl Personen. Dies können verschiedene Wesen sein. In Eidolon gebietet der Spieler nur über Menschen, doch der Computer steuert Mönche, Untote, Skelette, Drachen und weitere Monster. In anderen Varianten von Atlantis gibt es unterschiedliche Lebewesen, die ein Spieler kontrollieren kann (z.B. Elfen, Gnome, etc.). Möglicherweise kann eine Einheit auch aus verschiedenen Rassen gemischt sein. Um flexibel zu sein, werden die Personen als "Gegenstand" betrachtet (siehe nächsten Abschnitt).

Gegenstände (Item)

Gegenstände (*Item*) können sowohl einer Einheit gehören als auch in einer Region vorhanden sein, ohne einer Einheit zugeordnet zu sein. Es ist wünschenswert, die Gegenstände zu kategorisieren, nicht nur nach Zugehörigkeit zu Region oder Einheit, sondern noch detaillierter. Dies wird über ein Bitfeld (*Type*) realisiert, das folgende Kategorien kennt:

ItemUnit Gegenstand kann einer Einheit gehören.

ItemRegion Gegenstand kann direkt einer Region zugeordnet sein.

ItemAbstract Keine eigentlichen Gegenstände, zum Beispiel Wirtschaftsdaten einer Region.

ItemResource Gegenstand ist eine Ressource.

ItemPerson Auch wenn Personen als Gegenstände betrachtet werden, sollten sie doch von normalen Gegenständen unterscheidbar sein.

[2]Dies wurde zur Optimierung nicht genau so umgesetzt. Siehe dazu Abschnitt 5.2.

ItemTrade Handelsgut, ein spezieller Typ von Gegenständen, die nur dem Kaufen und Verkaufen Zwecks Geldgewinn dienen.

ItemWeapon Gegenstand ist eine Waffe oder Rüstung.

ItemMagic Gegenstand ist magisch.

Durch die Umsetzung dieser Kategorien als Bitfeld kann ein Gegenstand theoretisch allen Kategorien angehören. Das macht keinen Sinn, aber es ist wichtig, dass alle Kombinationen möglich sind.

Talente (Skill)

Die Einheiten können Talente (*Skill*) erlernen. Je länger die Einheit lernt, desto besser beherrscht sie ein Talent. Allerdings steigt die Fähigkeit für eine Aufgabe nicht linear mit der Lernzeit. Je besser sie schon ist, desto länger muss die Einheit lernen, bis sie eine höhere Stufe (*Level*[3]) erreicht. Die Lernzeit wird in Tagen gemessen (*Days*).

Talente haben einen Namen (*Name*) und eine Beschreibung (*Description*), die von der Stufe abhängen kann. Sie haben im Spiel keine Nummern, doch werden sie hier dem bekanntwerden nach numeriert, um einen Primärschlüssel zu erhalten (*Nr*, zusammen mit der Stufe).

Die gelernten Zaubersprüche werden in der Anleitung nicht als Talente bezeichnet. Ausser dass sie keine Lerntage brauchen, haben sie aber die selben Attribute wie normale Talente und werden deshalb auch als solche behandelt.

Gebäude (Static) und Schiffe (Moving)

Gebäude und Schiffe sind beides Strukturen und haben einige Gemeinsamkeiten. Das zu verwendene Datenbanksystem erlaubt aber keine Generalisierungen, weshalb hier auch keine solche vorgesehen wird. Da Schiffe und Gebäude vom Spielserver getrennt nummeriert werden, wäre die Nummer zudem kein eindeutiger Primärschlüssel.

Beiden gemeinsam sind die Attribute Nummer (*Nr*), Name (*Name*), Beschreibung (*Description*), besitzende Einheit (*OwUnit*) und die Partei dieser Einheit (*OwFaction*).

Gebäude haben zudem eine Grösse (*Extent*), sie können beliebig ausgebaut werden. Die einzigen Gebäude in Eidolon sind Burgen. Andere Spiele kennen auch andere Gebäudearten. Diese können in dieser Struktur relativ einfach hinzugefügt werden. Die Entität heisst Static, weil sie auch andere statische Strukuren repräsentieren könnte.

Schiffe können nicht beliebig ausgebaut werden, sondern man gibt bei Baubeginn den gewünschten Typ (*Type*) an. Danach ist die mögliche Grösse bestimmt und das Schiff muss fertig gebaut werden, bis es benutzt werden kann. Dazu wird mit eine Prozentzahl (*Percent*) angegeben, wie weit es fertiggestellt wurde. Die Entität wurde ebenfalls deshalb Moving genannt, weil sie auch andere bewegliche Strukturen wie Zeppeline oder Ballone repräsentieren kann.

[3] Aus technischen Gründen heisst das Feld in der Datenbank nicht Level. Siehe Abschnitt 5.2.

Partei (Faction)

Jeder Spieler führt eine Partei und jede Einheit gehört zu einer dieser Parteien. Wenn die Partei einer Einheit nicht bekannt ist, wird die Parteinummer -1 verwendet. Die Parteien sind im Spiel nummeriert (*Nr*) und haben einen Namen (*Name*).

Eine Partei kann mit einer anderen alliiert sein. Um dies darzustellen, erhält die Partei noch ein Attribut Beziehung (*Attitude*), das neben "alliiert" die Werte "eigene Partei" und "keine spezielle Beziehung" annehmen kann. Einige Varianten von Atlantis lassen kompliziertere Einstellungen der Beziehung zu, was mit diesem Schema ebenfalls erfasst werden kann.

Geländeart (Terrain)

Auch die Geländeart enthält nicht viele Informationen, in der Auswertung steht nur ein Name (*Name*). Doch könnten andere Atlantisvarianten weitere Informationen zu einer Region haben, zum Beispiel wie viele Bewegungspunkte ihre Durchquerung kostet[4].

Als Primärschlüssel erhalten auch die Geländetypen eine willkürliche Nummer (*Nr*). Für die Darstellung ist ein weiteres Feld vorgesehen, das die Farbe (*Color*) für eine Region dieser Geländeart angiebt, im Format der Windows Farben[5].

Meldungen

Es gibt einige Typen von Meldungen. Alle Meldungen bestehen nur aus einem String, der einen Text für den Benutzer enthält. Der Typ ist wichtig, da Meldungen gewisser Typen unbedingt betrachtet werden sollten, andere aber nicht so wichtig sind.

Man könnte die Meldungen als eine Entität betrachten und ein Feld vorsehen für den Typ. Die andere Möglichkeit ist, jeden Typ als eigene Entität zu betrachten. Hier wird die zweite Variante verfolgt. Die Namen sind in Abbildung 5.1 alle aufgeführt.

Hilfstabellen für das Programm

Für VisEiA werden zudem einige Tabellen vorgesehen, die nicht mit den Daten direkt zu tun haben.

- Settings ist eine Tabelle, die einer Nummer eine andere Zahl zuordnet.
- StringSettings ist eine Tabelle, die einer Nummer einen Text zuordnet.
- SavedSkillStats dient dem Speichern von Statistikabfragen, die sich auf Talente beziehen.
- SavedItemStats dient dem Speichern Statistikabfragen, die sich auf Gegenstände beziehen.

Die Tabellen für Statistikabfragen werden im Abschnitt 29 besprochen.

[4]Tatsächlich haben einige neuere Atlantis Spiele solche Erweiterungen. In der vorliegenden Version wurde kein Feld dafür vorgesehen, doch können die Plugins eine Ergänzungstabelle für Geländarten anlegen und dort die Bewegungskosten oder andere Informationen speichern.

[5]Als hexadezimale Zahl ist die Farbe zusammengesetzt aus BBGGRR.

5.1.2 Beziehungen

Die Beziehungen gehen alle von einer der drei Entitäten Runde, Region und Einheit aus. Sie werden ausgehend von diesen drei analysiert.

Runde

Der Server sendet jede Runde alle Daten, die gerade bekannt sind. Manche Informationen wie die Koordinaten einer Region oder der Standort einer Burg ändern sich aber nicht, werden also jedes mal wiederholt.

Es gibt also Informationen, die nur für eine bestimmte Runde gültig sind und andere, die zeitlos sind.[6] Mit geringem Informationsverlust kann man alle Entitäten ausser den Meldungen als zeitlos betrachten und die Beziehungen als zeitgebunden. Ausnahme von konstanten Beziehungen sind die Parteizugehörigkeit der Einheit, der Geländetyp einer Region und in welcher Region eine Burg steht.

Direkte Beziehung haben die verschiedenen Meldungstypen.

Region

In einer Region hat es Einheiten, Gegenstände, Schiffe und Burgen. Zu Einheit, Gegenstand und Schiff muss als Attribut die Runde bekannt sein, für Gegenstand zudem noch die Anzahl. Die Burgen können sich, wie im vorherigen Abschnitt erwähnt, nicht bewegen und die Beziehung braucht deshalb keine Attribute.

Zudem ist der Region eine Geländeart zugeordnet, ebenfalls ohne weitere Attribute.

Einheit

Eine Einheit besitzt Gegenstände, kennt Talente, befindet sich in einer Burg oder auf einem Schiff und gehört einer Partei an. Die Beziehung zu Gegenstand braucht als Attribute die Runde und die Anzahl. Der Aufenthalt in einer Burg oder auf einem Schiff ist ebenfalls an eine Runde gebunden. Zur Vereinfachung hat die Beziehung zu den Talenten nur die Stufe und die Lerntage als Attribute, nicht aber die Runde. Damit kann der Lernfortschritt nicht mehr rekonstruiert werden, was aber auch nicht wichtig ist. Die Zugehörigkeit zu einer Partei ist fix.

[6] Wenn man strikt wäre, könnten nicht einmal die Namen der Regionen als zeitlos behandelt werden, da sie vom Spieler geändert werden können. Einerseits passiert das sehr selten, andererseits bedeutet das Ignorieren dieses Falles bloss, dass man nicht herausfinden kann, ob eine Region früher einmal anders hiess. (Die Koordinaten können sich ja nicht verschieben, denn sie sind der Primärschlüssel.)

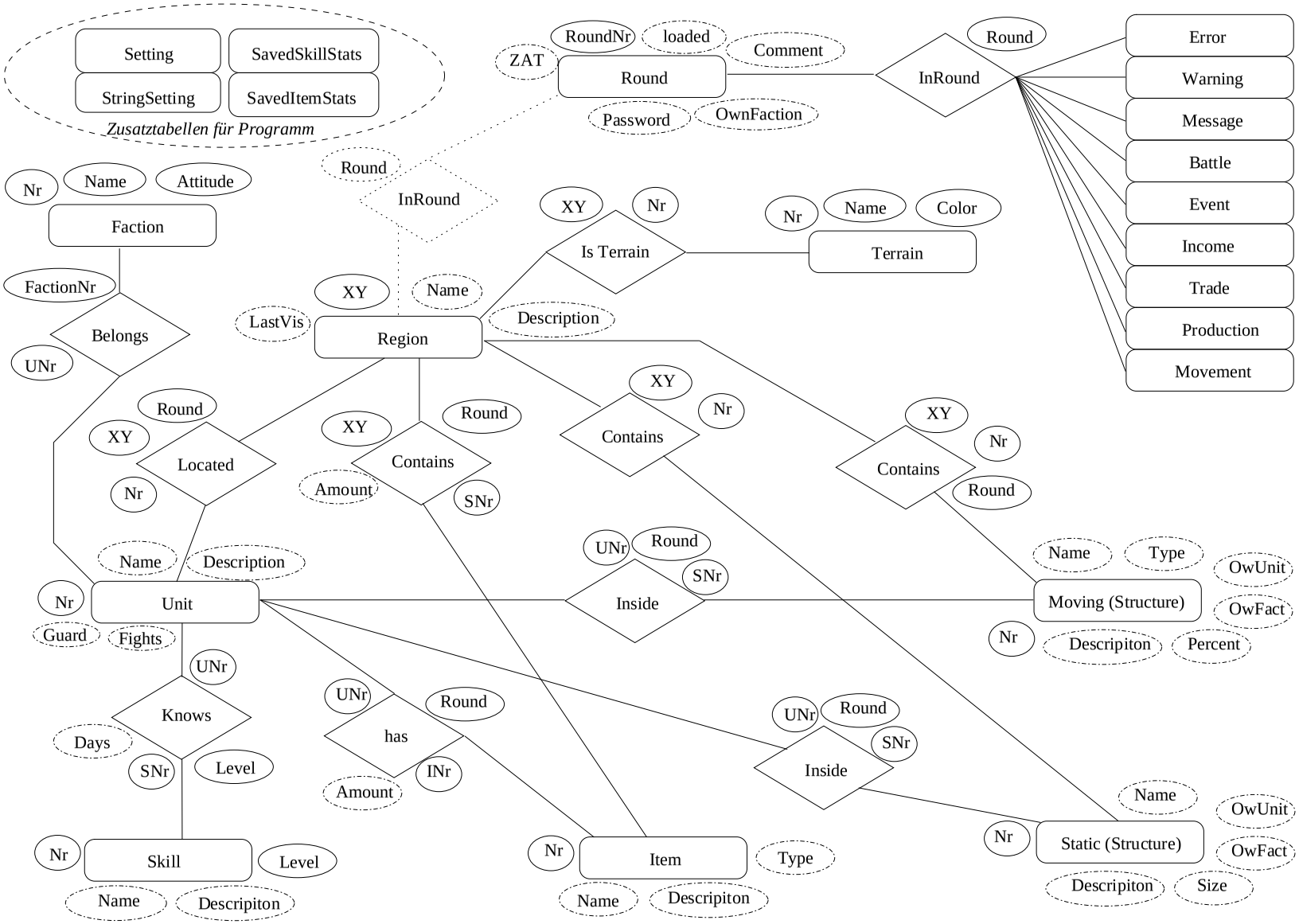


Abbildung 5.1: Die Datenstruktur von VisEiA, dargestellt im Entitäten-Beziehungsmodell

5.2 Abbildung auf die Datenbank

Mit dem Entitäten-Beziehungmodell aus Abbildung 5.1 lässt sich relativ einfach ein Datenbankschema bilden. Alle Entitäten werden als Tabellen definiert. 1 zu n Beziehungen werden den Tabellen zugeordnet, die die Entitäten enthalten, die genau einer Entität der anderen Tabelle zugeordnet sind. m zu n Beziehungen werden in eigenen Tabellen dargestellt.

Da die Koordinaten einer Region sehr häufig verwendet werden, lohnt sich hier eine Optimierung. Die beiden Felder X und Y geben einen zusammengesetzten Primärschlüssel. Weil die Dimensionen der Karte beschränkt sind, reicht *ein* INTEGER Feld für beide Koordinaten. Statt zwei Koordinaten gibt es deshalb das Feld XY, bei dem ein Teil der Bits als X, ein Teil als Y betrachtet werden. Damit kann eine Region über eine einzige Nummer eindeutig identifiziert werden.

Es folgt eine Auflistung der Tabellen mit ihren Attributen, (Primärschlüssel sind fett)[7]:

- *Round*: **RoundNr**, loaded, ZAT, Pwd[8], Comment, ownfaction
- *Terrain*: **Nr**, Name , Color
- *Faction*: **Nr**, Name, Attitude
- *Skill*: **Nr, Degree**, Name, Description , Icon
- *Item*: **Nr**, Name, Description , Type, Icon
- *Moving*: **Nr**, Name, Description , Typ, OwUnit, OwFaction, Percent, Icon
- *Static*: **Nr**, Name, Description , OwUnit, OwFaction, Extent, InRegion, Icon
- *Region*: **XY**, Name, Description , LastVis, isTerrain
- *Unit*: **Nr**, Name, Description , Faction, guard, fights, defOrder
- *RegionHasItem*: **XY, INr, Round**, Amount
- *RegionContainsMoving*: **XY, MNr, Round**
- *UnitInRegion*: **XY, UNr, Round**
- *UnitHasItem*: **UNr, INr, Round**, Amount
- *UnitKnowsSkill*: **UNr, SNr**, Degree[9], Days
- *UnitInMoving*: **UNr, MNr, Round**
- *UnitInStatic*: **UNr, SNr, Round**
- *Error*: **Round**, Info
- *Mess*[10]: **Round**, Info
  (restliche Meldungen analog)
- *Iconlist* **Nr**, Name, Bitmp

Für die graphische Oberfläche wird zudem ein Feld "Icon" hinzugefügt, das die Nummer des Icons bezeichnet, mit dem dieses Objekt dargestellt wird. Eine Tabelle mit Icons enthält die Nummern, Namen und im Feld "Bitmp" die Daten für ein Windows-Icon. Mit dieser Architektur kann ein Plugin spezielle Icons für seine Atlantisvariante einfügen. Andererseits kann der Spieler für alle Objekte ein persönliches Icon auswählen.

[7] Die genaue Beschreibung mit SQL findet sich im Quellcode, im Verzeichnis `LocalSQL`.
[8] Password ist in SQL ein reserviertes Wort, deshalb wird vom Modell abgewichen
[9] Level ist reserviert
[10] Message ist reserviert.

5.3 Klassen für die Spieldaten

Die Klassen können in Anlehnung an das ER Modell aus Abbildung 5.1 erstellt werden. Die Attribute werden mit Methoden abgefragt oder gesetzt, die mit get... und set... anfangen, z.B. `string getName()` und `setName(string n)`. Die Objekte werden durch Pointer miteinander verbunden. Übergeordneten Objekte entfernen bei ihrer Zerstörung auch gleich die Untergeordneten. Für die n/m Beziehungen werden Listen gebildet.

Als Basis für alle Klassen dient `VisObject`, die die Eigenschaften Namen und Nummer einführt.

Die Klasse `VisRound` enthält Listen, je eine für Regionen, Geländetypen, Parteien, existierende Fähigkeiten und Gegenstände, sowie für die verschiedenen Meldungstypen.

Jede `VisRegion` enthält Listen mit Pointern auf Einheiten, die sich in ihr aufhalten, auf Gebäude, die in ihr stehen und auf Schiffe. Die Gegenstände, die nicht den Einheiten zugeordnet sind, werden in einer doppelten Liste mit Nummer und Menge geführt. Weiter hat sie einen Pointer auf ihren Geländetyp.

Die `VisUnit` hat eine doppelte Liste mit Nummern ihrer Fähigkeiten und Gegenstände, mit der zugeordneten Menge beziehungsweise Lernzeit. Vorgesehen ist weiter ein Pointer auf die Struktur, falls sie in einer sind sind.

Die Strukuren werden mit einer Basisklasse `VisStructure` für die gemeinsamen Eigenschaften gebildet. Die Einheit und die Partei werden als Nummer und nicht als Pointer auf die Struktur gebildet, um die Instanzierung zu vereinfachen. Davon werden die Klassen `VisMoving` und `VisStatic` abgeleitet, die die restlichen Eigenschaften abbilden.

Die restlichen Klassen haben keine Referenzen auf andere Objekte, sondern nur Eigenschaften.

5.4 Aufteilung der Funktionalität

Das Programm wird in mehrere Bereiche zerteilt, die über Funktions- oder Methodenaufrufe miteinander kommunizieren. Die Schnittstellen sollen möglichst ANSI C++ kompatibel sein; so könnte ein einzelner Teil mit einer anderen C++ Umgebung geschrieben werden. Ein Datenmodul soll die Spieldaten in die Datenbank füllen und wieder herauslesen können, sowie Abfragen auf die Datenbank erlauben. Zur Wartung der Datenbank soll das Datenmodul zudem weitere Funktionen anbieten.

Die graphische Benutzerschnittstelle (GUI, graphical user interface) setzt auf den anderen Modulen auf und bietet ein zeitgemässes Arbeiten mit der Maus oder einem anderen Zeigegerät. Die wichtigsten Komponenten des GUI sind eine Weltkarte, Auswahl und Darstellung von Suchabfragen und Statistiken sowie die Verwaltung von Spielen.

Ein weiterer Teil ist die Funktionalität zum Einlesen von Auswertungen und dem Erstellen neuer Befehle. Dieser Teil sollte möglichst flexibel sein, um bei Änderungen des Spielservers einfach angepasst werden zu können.

Alle Teile arbeiten mit der gemeinsamen Klassenhierarchie aus Kapitel 5.3.

Die Struktur des Programmes wird in Abbildung 5.2 veranschaulicht.

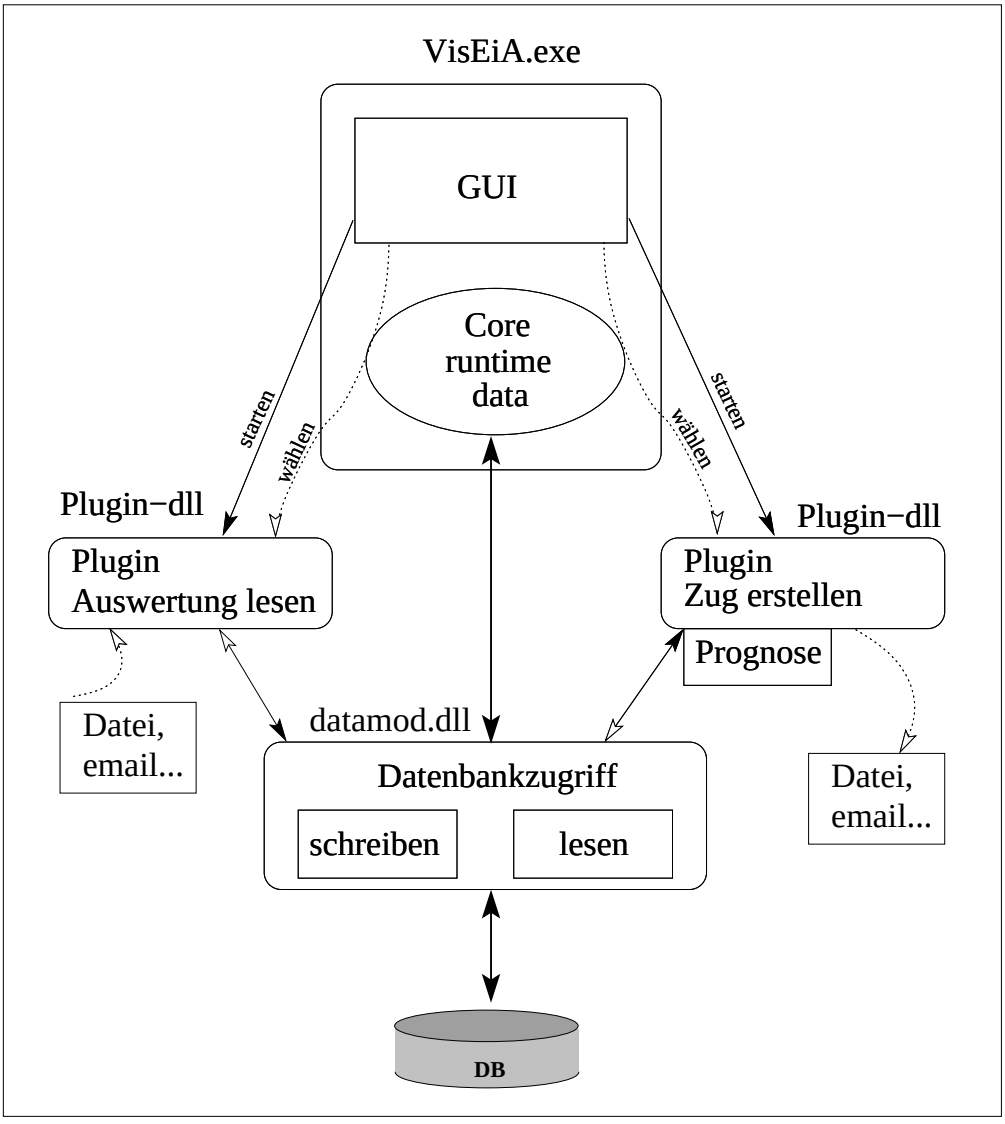


Abbildung 5.2: VisEiA setzt sich aus mehreren Teilen zusammen. Die Klassen für die Spieldaten werden für den Austausch zwischen ihnen verwendet.

5.4.1 Fehlertypen

Um eine sinnvolle Fehlerbehandlung zu ermöglichen, wird eine Hierarchie von Fehlerklassen erstellt. Das Grundobjekt ist der `VisError`, der von der C++ Klasse `exception` abgeleitet ist. Von VisError sind alle anderen VisEiA Fehlerklassen abgeleitet. Jeder Fehler enthält einen Titel für das Fehlermeldungsfenster, eine Information für den Anwender, eine technische Information für den Programmierer sowie den Namen der Methode, in der der Fehler auftrat.

Es gibt folgende Klassen von Fehlern, die alle direkt von VisError abgeleitet sind:

VisError title, userInfo, detailInfo, location

VisWrongStateException location, userInfo, State

VisDBError location, userInfo, detail

VisParserError location, filename, userInfo, detailInfo

VisFileError location, filename, detailInfo

VisFileNotFoundError location, filename

VisConfigurationError location, userInfo, detailInfo

VisUserError title, userInfo, detailInfo, location

Je nach Fehlerklasse werden die Fehlermeldungen direkt durch den Konstruktor festgelegt, unter zuhilfnahme der Parameter. `WrongStateException` und `VisDBError` können im Datenmodul auftreten, `ParserError` in einem Parser-Plugin, wie auch üblicherweise `VisFileError` und `VisFileNotFoundError`. `ConfigurationError` tritt auf, wenn die Konfiguration nicht stimmt, z.B. ein Registryeintrag nicht vorhanden ist. `UserError` zeigt einen Fehler des Benutzers an, eine Suchabfrage ohne Kriterien beispielsweise.

5.4.2 Datenmodul

Das Datenmodul erfüllt alle Aufgaben, die mit der Datenbank zu tun haben. Neben dem Speichern und Laden von Datenstrukturen für eine bestimmte Runde sind das Abfragen für die Einheitensuche und die Statistik. Schlussendlich sollen auch Funktionen zur Verwaltung der Datenbank vorhanden sein.

Um Datenbankabfragen optimieren zu können, hat das Datenmodul verschiedene Zustände, die durch Methodenaufrufe begonnen und wieder beendet werden (Siehe Abbildung 5.3).

Wenn das Datenmodul erzeugt wird, ist es im Zustand *init*. In diesem Zustand ist es nicht mit der Datenbank verbunden und akzeptiert keine Methodenaufrufe. Am Ende des Konstruktor wechselt es in den Zustand *none*. Dies ist der Normalzustand. Die restlichen drei Zustände *read, insert* und *setup* werden durch eine Methode init... begonnen und durch finish... beendet. Man kann nur aus dem Zustand *none* in einen anderen Zustand gehen, wird z.B. `initRead()` im Zustand *Insert* aufgerufen, ist das ein Fehler.

Die anderen Programme müssen sicherstellen, dass sie auch nach aufgetretenen Fehlern den Zustand korrekt beenden. Die Methode `getState()` kann dabei helfen, sie retourniert den Zustand, als einen Wert der Aufzählung `StateInit, StateNone, StateSetup, StateRead, StateInsert`.

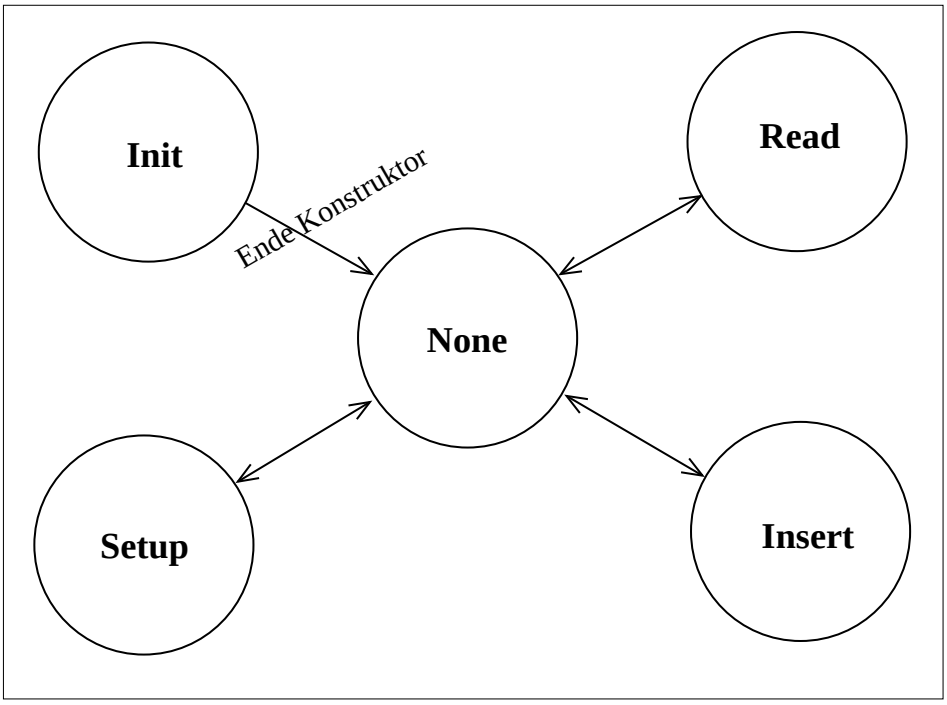


Abbildung 5.3: Die möglichen Zustände des Datenmoduls

Weil verschiedene Datenbanken unterschiedliche SQL Syntax haben, soll versucht werden, die SQL-Anweisungen in Dateien auszulagern.

Spieldaten speichern

Die wichtigste Prozedur ist `bool addRound(VisRound* round)`. Sie soll eine vollständige Runde in die Datenbank übertragen Dazu geht sie die Hierarchie durch und überträgt die Daten jedes Objekts in die Datenbank. Wenn ein Fehler auftritt, retourniert sie `false`, bei Erfolg `true`.

Weitere Prozeduren sollen es erlauben, die Listen von Talenten, Gegenständen, Parteien und Geländetypen zu ergänzen, ohne eine Runde einzulesen. Mehr für die interne Verwendung gedacht, kann auch eine Regionsliste für eine bestimmte Runde gespeichert werden.

```
bool updateAllSkill(VisAllSkill* lskill);
bool updateAllItem(VisAllItem* litem);
bool updateAllFaction(VisAllFaction* lfaction);
bool updateAllTerrain(VisAllTerrain* lterrain);
bool updateAllRegion(VisAllRegion* lregion, VisDate round);
```

Falls ein Objekt schon einen Eintrag in der Datenbank hat, soll dieser Überschrieben werden. Dadurch kann beispielsweise eine Beschreibung nachträglich verändert werden.

Spieldaten auslesen

Die Hauptfunktion soll die Objektstruktur für ein Datum erstellen, mit Hilfe der Informationen in der Datenbank. Hilfsfunktionen, die sowohl intern wie auch von ausserhalb nützlich sind, sind die readAll... Funktionen. Ihr Argument ist ein Pointer auf eine instanzierte - normalerweise leere - Liste entsprechenden Typs. Sie wird mit den Daten der Datenbank gefüllt.

```
VisRound* getRound(VisDate roundDate);
void readAllSkill(VisAllSkill* );
void readAllItem(VisAllItem* );
void readAllTerrain(VisAllTerrain* );
void readAllFaction(VisAllFaction* );
void readAllRegion(VisRound* round, VisAllRegion* lregion);
```

Um einzelne ältere Daten herauszufinden, sollen zudem Abfragen zu einzelnen Regionen, Einheiten oder Schiffen für ein beliebiges Datum möglich sein.[11]

```
VisRegion* getRegionAt(int x, int y, VisDate round);
VisUnit* getUnitAt(int nr, VisDate round);
VisMoving* getMovStructAt(int nr, VisDate round);
```

Statistiken

Die get Funktionen erstellen aus den Angaben eine SQL Abfrage, die in der Variable qry gespeichert wird. Die save Funktionen speichern die Angaben in der Datenbank, damit sie mit load wieder geladen werden können. loadSaved füllt die TStrings Liste mit den Namen aller gespeicherten Abfragen.

```
void getItemStatistics(TQuery* qry, int roundFrom, int roundTo,
            bool groupRegion, int factionnr, vector<int>& vecRegions,
            vector<int>& vecItems, bool itemRegion, bool accumulate);
void saveItemStatistics(AnsiString name, int roundFrom, int roundTo,
            bool groupRegion, int factionnr, vector<int>& vecRegions,
            vector<int>& vecItems);
bool loadItemStatistics(AnsiString name, int& roundFrom, int& roundTo,
            bool& groupRegion, int& factionnr, vector<int>& vecRegions,
            vector<int>& vecItems);
void loadSavedItemStatList(TStrings* names);

void getSkillStatistics(TQuery* qry, int factionnr,
            vector<int>& vecRegions, int skillnr);
void saveSkillStatistics(AnsiString name, int factionnr,
            vector<int>& vecRegions, int skillnr);
bool loadSkillStatistics(AnsiString name, int& factionnr,
            vector<int>& vecRegions, int& skillnr);
void loadSavedSkillStatList(TStrings* names);
```

[11] Diese Funktionen sind in der vorliegenden Version *nicht* implementiert.

Die get... Funktionen erwarten einen Pointer auf die CBuilder-Klasse TQuery, um das Ergebnis der Abfrage in dieses Objekt zu speichern. Diese beiden Methoden sind damit nicht ANSI C++ (weil TQuery keine ANSI C++ Klasse ist). Der Grund ist, dass im GUI mit einer Klasse gearbeitet wird, die Statistiken darstellen kann und die Daten direkt aus einer TQuery beziehen kann, um Programmieraufwand zu sparen. Wegen einem Fehler dieser graphischen Klasse[12] muss die Instanz von TQuery als Parameter an das Datenmodul übergeben werden. Normalerweise würde die Instanz von TQuery im Datenmodul erzeugt und als Rückgabewert verwendet.

Suchabfragen

Die Suchabfragen werden nicht direkt unterstützt. Es kann im graphischen Modul eine Abfrage als String erstellt werden und mit Hilfe der Methode `customQuery(AnsiString statement)` des Datenmoduls ein Objekt der Klasse `TQuery` erhalten werden. Es ist nicht konsequent, dass Statistiken und Suchabfragen so unterschiedlich umgesetzt werden. Dafür werden so zwei verschiedene Ansätze ausprobiert.

Icons

Die Icons können von VisEiA oder durch ein Plugin aus einer Resource (meist die eigene DLL) oder aus einer Datei eingefügt werden. Daneben gibt es Methoden, um ein Icon nach Namen zu finden, oder den Namen einer Nummer.

Für das GUI ist die nicht ANSI C++ kompatible Methode `readAllIcon`, mit der die Icons der Reihe nach in eine `TImageList` eingelesen werden. Da es nicht möglich ist, Icons aus der Datenbank zu löschen, ist die Nummer immer gleich der Position in der Liste.

```
int addIcon(long resinst, string resname, string iconname);
int addIcon(string path, string iconname);
int getIconNr(string iconname);
string getIconName(int iconnr);

void readAllIcon(TImageList* iconList);
```

Verwaltung der Datenbank

`bool install()` dient dem Erzeugen einer Datenbank mit allen Tabellen. Dazu wird eine Textdatei mit SQL-Anweisungen verarbeitet. Die Datenbank wird noch keine Einträge enthalten.

`bool deleteDatabase()` löscht die Datenbank mit Anweisungen aus einer Textdatei (Typischerweise `DROP TABLE`).

`bool cleanDB(VisDate round)` soll alles aus der Datenbank entfernen, was älter als das Datum *round* ist. Informationen mit Datumsangaben sind Round, RegionHasItem,

[12] Wenn die TQuery nicht im gleichen Programm und damit gleichen Speicherbereich erzeugt wird, wie die Klasse zur graphischen Darstellung, gibt es Speicherfehler. Das Datenmodul ist eine eigene Bibliothek und verwendet einen anderen Speicherbereich. Die Ursache des Fehlers ist dem Autor nicht bekannt, mit anderen Klassen traten derartige Fehler nicht auf.

RegionContainsMoving, UnitInRegion, UnitHasItem, UnitInMoving, UnitInStatic sowie alle Meldungen.

`bool deleteRound(VisDate round)` entfernt alles aus der Datenbank, was genau das Datum *round* hat.

Auch die init... und finish... Methoden zum wechseln des Zustandes können an der Datenbank Änderungen vornehmen. Für umfangreiches Einfügen könnten zum Beispiel die Indexe entfernt werden und nach dem Einfügen neu erstellt werden.

5.4.3 Graphische Benutzeroberfläche

Die Benutzeroberfläche ist der Teil des Programmes, mit dem der Anwender als erstes in Kontakt kommt. Ob die Logik des Programmes dahinter gut ist, wird er mit der Zeit auch feststellen, doch ein schlechtes GUI würde die Arbeit von Anfang an erschweren. Ein GUI zu programmieren ist sehr aufwendig, es gibt unzählige Fehlermöglichkeiten. Der Benutzer des Programmes sollte die sinnlosesten Befehle geben können, ohne dass das Programm abstürzt.

Der Anwender soll mit dem GUI alle Funktionen des Programmes nutzen können. Zuviele Schaltflächen überfordern den Anwender. Ich verfolge hier die Maxime:

> As simple as possible,
> As complicated as nessecary!

Um die Gestaltung des GUI einfacher zu machen, wird es nochmals aufgeteilt. Die relativ komplizierten Teilfenster für Suchabfragen, Statistikabfragen und für die Weltkarte werden als eigenständige Teile entwickelt. Für die Abfragen eignet sich die Klasse `TFrame`. Ein Frame ist wie ein ganzes Fenster zum Entwickeln, kann aber am Schluss in ein anderes Fenster eingefügt werden. Für die Weltkarte wird eine CBuilder Komponente geschrieben (Komponenten sind spezielle Klassen, die in CBuilder graphisch manipuliert werden können).

Fensterkonzept

Wenn VisEiA gestartet wird, soll zuerst das Spiel ausgewählt werden. Ein Startfenster, das relativ schnell geladen ist, bietet die verfügbaren Spiele an und erlaubt es, ein neues Spiel zu erstellen. Wenn der Benutzer ein Spiel lädt, wird das Hauptfenster mit dem gewählten Spiel gezeigt.

Für das Erstellen eines neuen Spiels gibt es einen eigenen Dialog, wo der Name für das Spiel vergeben wird, sowie der Speicherort für die Datenbank und die Plugins bestimmt werden. Eventuell sollte der Benutzer auch andere als die Standard SQL Anweisungen für das Einrichten der Datenbank angeben können. Falls die Angaben korrekt sind, wird der Dialog zum Einlesen einer Auswertung gezeigt, um dann das Hauptfenster gleich mit interessanten Daten anzuzeigen.

Der Dialog “Auswertung einlesen” fragt nur den Dateinamen ab. Die Logik zum Verarbeiten der Auswertung steckt im Plugin, das die Daten direkt in die Datenbank speichert.

Die Plugins müssen vor der Verwendung im Registrybereich von VisEiA registriert werden, was mit dem PluginBrowser gemacht wird. Dieses Fenster erlaubt das Anwählen

einer dll und überprüft, ob sie ein gültiges VisEiA Plugin ist. Falls sie es ist, kann man sie zu den verfügbaren Plugins hinzufügen.

Für die Konfiguration soll es einen einzigen Dialog geben, der alle möglichen Einstellungen erlaubt. Um die Daten getrennt von den Fenstern zu halten, gibt es eine Klasse Namens CoreData, die die Verbindung zur Datenbank verwaltet.

Eine weitere wichtige, nicht visuelle Klasse ist die ActionCentral, die Anforderungen von einem Fenster an ein anderes weitergibt. So soll die Trennung zwischen den Fensterklassen besser eingehalten werden. In Abbildung 5.4 wird die ActionCentral zur Vereinfachung nicht dargestellt. Alle Übergänge zwischen Fenstern finden in Wirklichkeit über sie statt und nicht direkt.

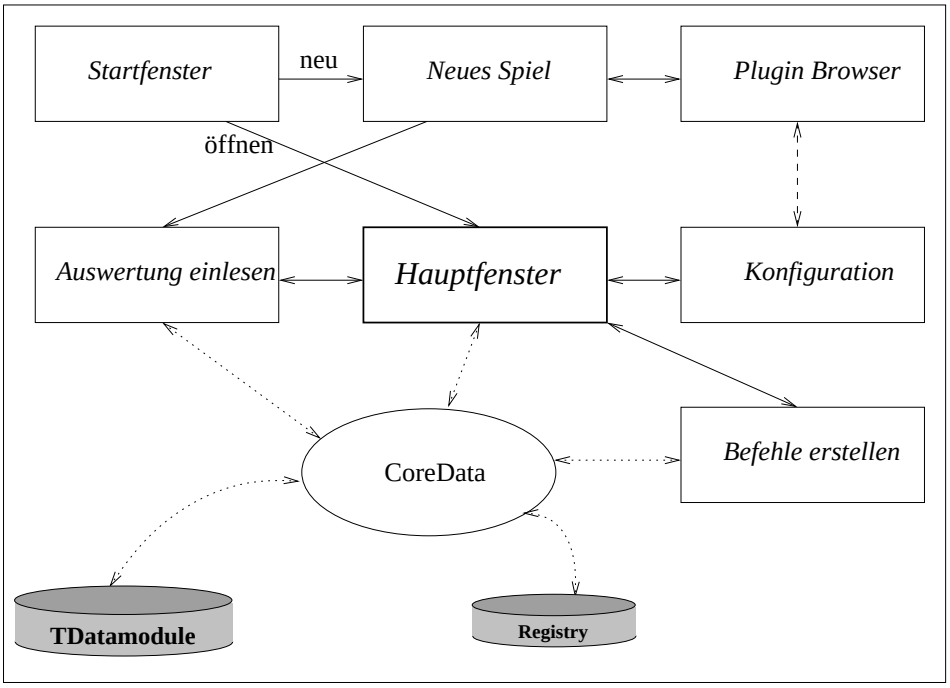


Abbildung 5.4: Konzept des GUI. Die Fenster sind durch Rechtecke dargestellt, CoreData ist eine Klasse ohne Fenster (als Ellipse dargestellt). Durchgezogene Linien bedeuten die Übergabe des Fokus zwischen den Fenstern, gepunktete die Abfrage von Daten.

Hauptfenster

Das Hauptfenster enthält ein Menü und eine Zeile mit Bildchen, die man anklicken kann, um häufig benötigte Funktionen aufzurufen (sogenannte Toolbar). Die eigentliche Fensterfläche wird in zwei Bereiche aufgeteilt.

Im schmaleren linken befindet sich die Steuerung, SearchCreator und StatisticCreator, sowie ein Baum, der die Regionen mit untergeordneten Elementen darstellt[13], ähnlich der Verzeichnisansicht des Windows-Dateimanagers. Bei Klick auf eine Region soll die Karte auf diese zentriert werden. Zwischen diesen drei Steuerungsbereichen soll mit einer `TTabbedPane` hin und her gewechselt werden können. Die TabbedPane ermöglicht es,

[13]Diese Idee wurde von EidolonC [21] abgeschaut.

mehrere Teilfenster übereinander zu definieren, die wie Registerkarten eine vorstehende Beschriftung haben.[14]

Der grössere rechte Bereich wird ebenfalls abwechslungsweise für mehrere Zwecke verwendet. Es beinhaltet die Karte, eine Tabelle mit den Ergebnissen der Suche, die Statistiken und eine Liste der Meldungen. Das Auswählen zwischen diesen drei wird ebenfalls mit einer `TTabbedPane` ermöglicht.

Wenn eine Suche ausgeführt oder eine Statistik erstellt wird (linker Teil), soll die entsprechende Seite im rechten Teil nach vorne gebracht werden.

Weltkarte

Da die Welt von Atlantis in quadratische Felder aufgeteilt ist, eignet sich als Basis für die Weltkarte die Komponente `TCustomGrid`. Als Grundlage für tabellarische Strukturen gedacht, erlaubt sie bereits Funktionen wie das Scrollen, wenn die Fläche nicht reicht, um alle Felder darzustellen. Von dieser Komponente wird unsere Komponente `TMapDrawGrid` abgeleitet.

Die Komponente soll an den Rändern die Koordinaten anschreiben und in den Feldern die zugeordnete Regionen darstellen. Die Regionen werden durch die Klasse `TMapField` repräsentiert. Diese Hilfsklasse erlaubt das Zeichnen von genau einem Feld. Standardmässig zeichnet sie das Feld weiss, löscht es also. Die Felder werden zugeordnet mit:

```
void setField(int x, int y, TMapField* field);
TMapField* getField(int x, int y);
```

TMapDrawGrid soll in der Lage sein, nur einen Teil des Koordinatensystems darzustellen, weil kaum ein Spieler die ganze Welt kennt. Dazu gibt es die Felder `MinCol`, `MaxCol`, `MinRow` und `MaxRow`.

Die Grösse der Felder kann eingestellt werden und die Komponente bietet eine Methode zum Zoomen sowie zum Zentrieren auf ein Koordinatenpaar. Die Felder `MinCellSize` und `MaxCellSize` erlauben es, Grenzen für die Feldgrösse anzugeben. Für `CellSize` kann eine beliebige Grösse angegeben werden, ausserhalb der Grenzen wird sie korrigiert. `ZoomStep` schliesslich definiert die Schrittgrösse, mit der `ZoomIn()` und `ZoomOut()` die Feldgrösse verändern.

Es soll möglich sein, in jeder Region die Anzahl dort vorhandener Gegenstände eines Typs anzuzeigen. Welche angezeigt werden können, kann vom Benutzer konfiguriert werden, ebenso ob sie im Moment angezeigt werden sollen. Um diesen Mechanismus zu unterstützen, gibt es in TMapDrawGrid die Aufzählung `DrawOpt`, deren Elemente in eine Menge namens `DrawOptions` eingefügt werden können. Jedes Element, das in der Menge ist, soll dargestellt werden. (Die Umwandlung von der Zahl zu einem VisItem erfolgt über `TCoreData`.)

Um die Karte vor dem Laden eines neuen Spiels zurückzusetzen, hat die Karte noch eine Methode `resetGrid()`.

[14] Abbildung 7.2 zeigt eine solches Karteikartensystem.

Suchabfragen erzeugen

Die Abfragen sollen die Beantwortung einiger wichtigen Fragestellungen für das Spiel Atlantis erlauben. So ist es immer wieder wichtig zu wissen, wo sich eine Einheit mit einer bestimmten Fähigkeit befindet, oder wer eine genügende Menge eines Gegenstandes besitzt. Interessant ist auch die Frage, in welcher Region es noch genügend Rohstoffe hat oder eine grosse Bevölkerung.

Um solche Abfragen eingeben zu können, wird die Klasse `TSearchCreator` entwickelt. Da nur über die Regionen Daten vorhanden sind, in denen sich Einheiten befinden, reicht es, wenn man auf Fragen zu Regionen als Ergebnis die Einheiten erhält, die sich in dieser Region aufhalten.

Für die Regionen kann als Einschränkung ein Bereich von Koordinaten angeben werden und beliebig viele Kriterien, dass die Region mindestens oder höchstens eine bestimmte Anzahl Gegenstände enthalten muss.

Für die Einheit kann man die Einheitsnummer angeben, womit nur eine Einheit gefunden wird. Weiter kann man Gegenstände, die eine Einheit haben kann, verlangen und auch Fähigkeiten verlangen. Zudem kann man angeben, dass nur Einheiten einer bestimmten Partei gefunden werden sollen.

Das Ergebnis der Suche wird einer Tabelle vom Typ `TDBGrid` zugeordnet, die die Datenzeilen als Tabellenzeilen darstellt. Es sollte möglich sein, durch anklicken einer Zeile diese Einheit im Regionenbaum anzuzeigen und die Karte auf die entsprechende Region zu zentrieren.[15]

Die Suchabfragen sollen gespeichert und nach erneutem Programmstart wieder geladen werden können.

Statistikabfragen erzeugen

Die Statistiken werden konfiguriert mit einem Formular der Klasse `TStatisticCreator`. Es sind Statistiken über Gegenstände oder über eine Fähigkeit möglich. Der Benutzer kann die Regionen anwählen, die in der Statistik berücksichtigt werden.

Wenn eine Statistik über Gegenstände erstellt wird, kann der Zeitraum gewählt werden. Die verschiedenen Gegenstände werden pro Runde summiert. Man kann auswählen, ob die Statistik den über alle Regionen aufsummierten Verlauf zeigen soll, oder die Kurve jeder Region einzeln.

Für Fähigkeiten macht das keinen Sinn, weil sie nur für die aktuelle Runde in der Datenbank vorliegen. Es kann nur von einem Talent gezeigt werden, in welchen Regionen es im Maximum wie gut bekannt ist.

Schlussendlich kann man die Partei wählen, von der man die Statistik sehen möchte.

Eine andere Möglichkeit ist der Vergleich der Anzahl eines bestimmten Gegenstandes zwischen zwei Parteien[16].

Zur Darstellung der Statistik kann das Packet teeChart[17] verwendet werden. Es erlaubt als Datenquelle für ein Diagramm direkt ein Objekt vom Typ `TQuery`.

[15] Diese Funktionalität konnte leider nicht implementiert werden, weil die Zeit nicht reichte.

[16] Leider konnte diese Funktion aus Zeitgründen nicht implementiert werden

[17] Packet zum Erstellen von Diagrammen, das in der Basic-Version mit CBuilder mitgeliefert wird. Entwickelt von David Berneda, `http://www.teemach.com`

5.5 Kommunikation mit dem Spielserver

Von Atlantis gibt es, wie in Kapitel 2 erwähnt, unterschiedliche Varianten. Sie haben leicht verschiedene Dateiformate für die Auswertungen und andere Syntax für die Befehle. (Dies schon deshalb, weil in den Varianten die Spielmöglichkeiten erweitert werden.) Trotzdem sind die Daten recht ähnlich, die Datenbank und die graphische Darstellung eignen sich für verschiedene Spiele. Damit von VisEiA selber nicht unterschiedliche Versionen für die Spielvarianten existieren müssen, sind die Programmteile, die Auswertungen einlesen oder bei der Befehlserstellung helfen, als Plugins implementiert. Ein Plugin ist eine Bibliothek, die während der Laufzeit dynamisch geladen wird. Das Hauptprogramm verwendet das Plugin durch Aufrufe von Funktionen, die dieses "Exportiert".

Somit kann für andere Spiele als Eidolon ein Plugin erstellt werden, das die Kommunikation mit dem Spielserver erlaubt. In Eidolon wird die Auswertung als Textdatei gemailt und die Befehle für den Zug in ein Email geschrieben. Es wäre denkbar, ein Plugin zu schreiben, das selber mit dem Mailserver Kontakt aufnimmt, oder sogar ein Spielserver, der permanent mit dem Internet verbunden ist und über ein eigenes Protokoll mit dem Plugin kommuniziert.

Schnittstellen für Plugins

Ein Plugin ist eigentlich nichts anderes als eine normale Programmbibliothek. Unter Windows bekommen diese die Endung .dll (für Dynamic Link Library). Damit VisEiA das Plugin verwenden kann, muss dieses einige Funktionen exportieren. Eine Gruppe von Funktionen dient dem erkennen, ob es sich um ein geeignetes Plugin handelt und der Information für den Benutzer. Für die Aufgaben "Auswertung einlesen" und "Zug erstellen" gibt es jeweils einige weitere Funktionen, die die eigentliche Aufgabe ausführen.

Ist ein Plugin geladen, so kann es seinerseits die Funktionalität des Datenmoduls verwenden.

Die Funktionen, die alle VisEiA Plugins haben müssen, sind

```
extern "C" int getVisEiAPluginType();
extern "C" char * getPluginName();
extern "C" char * getPluginInfo();
extern "C" char * getPluginVersion();
extern "C" bool installPlugin(Datamodul*);
extern "C" bool updatePlugin(Datamodul*);
```

Wenn das Plugin Auswertungen einlesen kann, so muss es als Type den Wert 1 retournieren, falls es zum Erstellen der Befehle ist, den Wert 2. Der Name sollte kurz und informativ sein, damit der Benutzer weiss, ob das das richtige Plugin ist. Die Info darf länger sein und kann zum Beispiel Informationen über den Autor oder die Spielhomepage enthalten. Die Version soll dem Benutzer helfen, wenn mehrere Versionen eines Plugins existieren.

Wenn ein Spiel erstellt wird, so werden auch die Plugins gewählt. Nach dem Erstellen der Datenbank wird für das Plugin die `installPlugin` Methode aufgerufen. Das Plugin sollte die Datenbank für sein Spiel vorbereiten und kann Vorbereitungen treffen, um selber Daten zu speichern.

Wenn zu einer neuen Version eines bereits für die Datenbank gewählten Plugins gewechselt wird, ruft VisEiA dessen Methode `updatePlugin` auf, damit es die Möglichkeit hat, den Zustand der Datenbank oder andere Informationen zu ändern.

Die Anweisung `extern ''C''` veranlasst den Compiler, diese Funktion zu exportieren. Der Qualifizierer "C" sorgt dafür, dass der Funktionsname genau so exportiert wird, wie er hier steht. Normalerweise würden den Funktionen nach C++ Stil Buchstaben angehängt, um die Argumentstypen und den Rückgabetyp anzugeben. Dann können sie aber nicht mehr über den normalen Namen geladen werden, weshalb sie eben im C Stil exportiert werden müssen.

Auswertung einlesen

Eidolon und andere Atlantis Varianten kennen zwei Typen der Auswertung: eine Menschenlesbare, die die Informationen lesbar formatiert und mit Beschriftungen der Werte präsentiert, sowie eine für den Computer geeignete mit einfacher Syntax und ohne überflüssigen Text. Man kann als Spieler wählen, welche der beiden Auswertungen man erhält oder ob man beide möchte.

Für den Spieler ist es einfacher, wenn sein Plugin die menschenlesbaren Auswertung verarbeitet, da standardmässig diese gesendet wird. Für den Programmierer ist es hingegen wesentlich bequemer, die Computerauswertung zu verarbeiten, da diese für eine einfache automatische Abarbeitung konzipiert wurde.

Die Funktionen des Einlese-Plugins sind

```
extern "C" bool withHumanReport();
extern "C" bool withComputerReport();
extern "C" bool readFile(char * Filename, Datamodul*);
```

Die ersten beiden Funktionen dienen dem Abfragen, ob das Plugin mit der menschenlesbaren bzw. der Computerauswertung etwas anfangen kann. Sinnvollerweise sollte mindestens eine der beiden Funktionen `true` retournieren.

Die letzte Funktion muss die eigentliche Funktionalität implementieren. Die Auswertung in der Datei muss eingelesen werden und mit Hilfe des Pointers auf das Datenmodul in dieses eingefüllt werden. Wenn das Einlesen gelingt, wird `true` retourniert, wenn es misslingt, `false`.

Das Plugin sollte dafür sorgen, dass alle Fehler aufgefangen werden. Wenn es im Plugin einen schweren Fehler gibt, kann das ganze Programm abstürzen. Einzig die Fehler `VisParserError`, `VisFileError` und `VisFileNotFoundError` dürfen geworfen werden.

Zug erstellen

Am Ende der Auswertung ist z.B. bei Eidolon eine Vorlage für die nächsten Befehle angefügt. Sie enthält alle Einheiten und gibt ihnen dieselben Befehle die sie in der letzten Runde erhalten haben.

Ein Plugin zum Erstellen eines neuen Zuges könnte diese Vorlage verwenden oder aus der Datenbank selber die eigenen Einheiten suchen und Befehle erfragen. Da der letzte Befehl in der Auswertung steht und in die Datenbank eingelesen wird, können auch mit der zweiten Variante sinnvolle Voreinstellungen angeboten werden.

Das Plugin könnte zudem eine Prognose machen, wie die Situation in der nächsten Runde aussieht. Dabei sollte es möglichst die neuen Befehle berücksichtigen. Der Spieler kann so überprüfen, wie sich seine Befehle auswirken werden und ob es Probleme geben wird. Überprüft werden könnte die Entwicklung der Rohstoffe oder der Nahrungsreserven.

```
extern "C" bool withTemplate()
extern "C" bool withoutTemplate()
extern "C" bool withPrognos()

extern "C" bool createTurn(Datamodul* dm)
extern "C" bool createTurnFromTemplate(char* templatefile, Datamodul* dm)

 //Prognose
extern "C" int RegionSilver(int x, int y)
extern "C" int UnitSilver(int unit)
extern "C" int RegionItems(int x, int y, int item)
extern "C" int UnitItem(int unit, int item)
extern "C" int UnitSkill(int unit, int skill)
extern "C" int UnitPositionX(int unit)
extern "C" int UnitPositionY(int unit)
```

Um die Befehle zu Erstellen, kann das Plugin eigene Fenster öffnen und darin die Informationen abfragen, zum Beispiel für Dateinamen.

Die Funktionalität der Prognose konnte aus Zeitgründen leider nicht mehr in die GUI implementiert werden und hat somit vorläufig keinen Sinn. Auch ist die Form möglicherweise nicht besonders praktisch.

Kapitel 6

Implementierung

Die Umsetzung der Ideen in ein lauffähiges Programm nahm sehr viel Zeit in Anspruch. Das Programm funktioniert erfreulicherweise, nur das Plugin zum Aufstellen von Befehlen oder für Prognosen konnte nicht realisiert werden.

In diesem Kapitel soll die Programmierung von VisEiA beschrieben werden. Zum besseren Verständnis empfiehlt es sich, neben diesem Dokument den Quellcode anzuschauen. Viele Details sind nur aus diesem ersichtlich. Die Abbildungen zu den Fenstern befinden sich im Kapitel 7, Bedienungsanleitung.

Die Gliederung der Abschnitte richtet sich nach der Aufteilung des Code auf Datenmodul, Hauptprogramm und Plugins. So sind zum Beispiel die Klassen für die Spieldaten in der Datenmodul-DLL.

6.1 Allgemein

In VisEiA wird stark Gebrauch gemacht von der STL[1]. Diese Bibliothek definiert nützliche Klassen und Algorithmen als Templates[2]. Hauptsächlich wurden die Klassen `vector` und `map` und die Suchalgorithmen für Vektoren benutzt.

Aus Hinweisen im Internet und der CBuilder Hilfe leiten sich einige für die Effizienz wichtige Fakten ab. In den `#include` Anweisungen müssen die Dateien der STL als erste aufgeführt werden, da sonst einige Deklarationen Mehrfach ausgeführt werden. Weiter empfiehlt es sich, folgende beiden Zeilen an den Anfang der Quelldateien zu stellen:

```
#define __MINMAX_DEFINED // use STL's generic min and max templates
#define __USE_STL // exclude BC++'s redundant operator definitions
```

Ein weiterer Hinweis aus einer Newsgroup[7] brachte eine starke Beschleunigung der Kompilierung mit sich. In jeder Datei, die Standardklassen verwendet, schliesst man am Anfang der Datei die Header Dateien der verwendeten Klassen ein. Insbesondere bei Klassen, die die VCL des CBuilder verwenden, sind das viele, grosse Dateien.

[1] Standard Template Library, Implementation von The Rogue Wave, wurde 1998 von der ANSI und der ISO akzeptiert. Die STL ist Bestandteil von CBuilder.

[2] Templates sind spezielle Deklarationen für Funktionen und Objekte, die sich nicht auf spezifische Klassen beziehen. Deshalb können sie auch auf selber definierte Klassen angewandt werden. Eine gute Quelle ist [2]

CBuilder kann die gelesenen Header Dateien zwischenspeichern, sie werden von Borland als präkompilierte Header bezeichnet. Erreicht wird das durch die Anweisung `#pragma hrdstop`, die das Ende der präkompilierten Header markiert. Dieser Mechanismus funktioniert nur, wenn die Dateien genau die gleichen Header einschliessen. Bei Unterschieden wird jede Konfiguration eingelesen und zwischengespeichert.

Weil es schwierig ist, die `#include` Anweisungen aller Dateien gleich zu halten, sollte man alle verwendeten Bibliotheks-Header in einer separaten Datei einschliessen, die mit `#pragma hrdstop` endet, und diese Datei in allen Quelldateien des Projekts am Anfang einschliessen.

6.2 Datenbankmodul

Diese DLL exportiert neben dem Datenmodul `TDatamodule` noch weitere Klassen und Funktionen, die in allen Programmteilen gebraucht werden. Daneben enthält sie Code, der nur zur internen Verwendung bestimmt ist.

Klassen für die Spieldaten

Die einfachen Listen wurden mit einem `vector` implementiert. Dieser erlaubt das Suchen nach einer Nummer oder einem Namen.

Weil die Suche möglich ist, können die Listen von Gegenständen der Region und der Einheit mit Gegenstandsnummern gefüllt werden, anstatt mit den Pointern. Dies vereinfacht das Instanzieren der Objekte, bedeutet aber einen Effizienzverlust.

Die Koordinaten werden folgendermassen auf die Zahl aufgeteilt: Die hintersten 12 Bit sind die X Koordinaten, die mittleren 12 die Y Koordinaten und die letzten 8 werden nicht verwendet. Die Umrechnung wird mit folgenden Funktionen erreicht:

```
xyToC(int x, int y);
CxyToX(int c);
CxyToY(int c);
```

Das Bitfeld für die Gegenstände kann folgendermassen verwendet werden:

```
if (item.getType() && ItemResource) { // Gegenstand ist Resource
 ...
}
```

Datenbankzugriff

Die Hauptfunktionalität des Datenmodules wurde in der Klasse `TDatamodule` implementiert.

Daneben wurden einige Hilfsfunktionen geschrieben, zum Beispiel um die Spieldaten - Hierarchie in die Datenbank zu schreiben.

Zum Lesen einer Runde werden nicht Funktionen verwendet, sondern die Vis... Klassen wurden abgeleitet mit einem Konstruktor, der die Daten direkt aus der Datenbank liest.[3]

[3]Siehe zu diesem Ansatz auch Kapitel 8.2.

Statistik

Die Statistiken können über die in Abschnitt 20 beschriebenen Methoden erzeugt werden. Die Methoden übersetzen dabei die Argumente in ein SQL Statement, das für kompliziertere Abfragen sehr lang sein kann.

Einige Arbeit brauchte das Speichern der Statistikabfragen. Da für die gewünschten Gegenstände und Regionen jeweils Vektoren angegeben werden, haben die Daten eine variable Länge. Um diese Listen zu speichern, wurde der Feldtyp "Blob"[4] gewählt, ein Typ für grössere, unspezifizierte Datenmengen.

Ein Blobfeld kann mit einem `TStream` beschrieben werden und wieder in den `TStream` gelesen werden. Eine Hilfsfunktion wandelt einen int-Vektor um in einen Strom. Als erstes wird die Länge des Stroms hineingeschrieben, dann werden alle Werte hintereinander geschrieben. Damit kann der Strom einfach zurück verwandelt werden, in dem er in ein Array gelesen wird, das dann gleich die Zahlen wieder enthält.

Es war nicht das Ziel, die erzeugte SQL Abfrage zu speichern. Mit der hier gewählten Lösung ist es nähmlich möglich, im Statistikformular den ganzen gespeicherten Zustand wieder herzustellen, damit ihn der Benutzer anpassen kann.

6.3 TMapDrawGrid

`TMapDrawGrid` wurde als CBuilder Komponente implementiert. Sie kann in der Entwurfsansicht des CBuilder graphisch manipuliert werden. Die Parameter erlauben alle Einstellungsmöglichkeiten, die in Abschnitt 23 definiert wurden.

Weil `TCustomGrid` die Zuordnung von Objekten zu den einzelnen Feldern nicht unterstützt, wir aber die Regionen in einer objektorientierten Struktur haben, wurde ein zweidimensionaler Vektor (Vektor von Vektoren) geschaffen, der für jede Region ein Objekt enthält. Dieses Objekt muss abgeleitet sein von der Klasse `TMapField`. Sinnvollerweise enthält die Feldklasse Daten, die sie für die Darstellung braucht. Die wichtigste Methode des Feldes ist `draw`, in der es sich selber auf die Bildschirmregion Rect zeichnen muss.

```
void draw(TMapDrawGrid* MapDrawGrid, const Windows::TRect &Rect,
          TGridDrawState State)
```

6.4 GUI

Das graphische Interface konnte auf die Vorarbeit aufbauen. Die Fenster wurden weitgehend aus Standardobjekten von CBuilder zusammengesetzt, mit Ausnahme der erwähnten Karte.

MainForm

Das Hauptfenster konnte wie geplant realisiert werden. Es hat zwei Toolbars, die sich den oberen Fensterbereich teilen: eine fixe Toolbar mit den wichtigsten Befehlen und

[4] Blob steht für **B**inary **L**arge **Ob**ject.

eine dynamische, auf der die möglichen Angaben zu Gegenständen in den Regionen ein- oder ausgeblendet werden können. Letztere muss dynamisch sein, weil erst zur Laufzeit bekannt ist, welche Gegenstände der Benutzer eventuell sehen will.

Plugin Browser

Der Plugin Browser stellt einen Verzeichnisbaum und den Inhalt des gewählten Verzweichnisses dar. Wenn man eine Datei mit der Endung dll anklickt, versucht er sie als VisEiA Plugin zu laden. Wenn das gelingt, listet er die Informationen auf und man kann das Plugin hinzufügen. Durch die Unterscheidung der Plugin - Typen kann der Manager den Typ selber entscheiden.

Die Plugins werden in der Registry gespeichert, weil zu diesem Zeitpunkt noch keine Datenbank verfügbar ist.

Suchabfragen

Das Formular wurde als `TFrame` definiert. Damit kann es einfach in das Hauptfenster integriert werden. Es erlaubt die Eingabe von Suchkriterien für die Region, für die Einheit, für die Anzahl Gegenstände und für die Talente. Die Eingabefelder werden dynamisch im Code erzeugt. Immer wenn man ein Kriterium eingegeben hat, wird ein neues Feld erzeugt, um ein weiteres Kriterium einzugeben. Die Felder werden in internen Listen verwaltet und bei Ausführen der Suche durchgegangen.

Weil das Datenmodul keine speziellen Methoden für die Suchabfragen anbietet, muss die Abfrage hier in die SQL Sprache gebracht werden. Die Name wird mit LIKE verglichen, was auch ungenaue Abfragen erlaubt. Das Jokerzeichen % steht für eine beliebige Zeichenfolge, der Tiefstrich _ für genau eines.

Das erzeugte SQL wird mit der Datenmodul-Methode `customSelect` ausgeführt. Die erhaltene `TQuery` wird direkt einer `TDBGrid` als Datenquelle zugewiesen. Diese stellt die Abfrage als Tabelle dar.

Eine kleine Verbesserungsmöglichkeit wäre, bei Doppelklick auf eine Reihe zu diesem Feld auf der Karte zu springen. Leider konnte das nicht mehr implementiert werden.

Statistiken

Auch für die Statistiken wurde ein `TFrame` definiert. Die Auswahl der Regionen erfolgt aus einer Liste, die Mehrfachauswahl zulässt. Mit einem Register kann man zwischen Gegenständen und Talenten auswählen. Gegenstände kann man ebenfalls mehrere aus einer Liste wählen. Einzig die Talente kann man nur einzeln betrachten. Da diese ja zudem nur für die aktuelle Runde gespeichert werden, kann somit nur der Vergleich eines Talentes zwischen den Regionen gemacht werden.

Der Vergleich zweier Parteien konnte leider nicht implementiert werden.

6.5 Plugin "Eidolon Computerauswertung"

Dieses Plugin liest eine Computerauswertung von Eidolon ein und speichert sie in die Datenbank. Die Auswertung ist in einer Textdatei, die vor der Computerauswertung

meist auch die menschenlesbare Version enthält. Eine spezielle Zeile markiert den Anfang der Computerauswertung. Pro Zeile hat es jeweils eine Information, manchmal in Anführungszeichen eingeschlossen.[5] Am Ende der Zeile kann es einen Kommentar haben, der mit einem Punktstrich beginnt.

Ein Objekt der Klasse `TokenReader` liest diese Datei ab der Markierungszeile ein und entfernt automatisch Kommentare und Anführungszeichen. Klassen, die von den Spieldaten - Klassen abgeleitet wurden, lesen in ihrem Konstruktor die Daten aus der Datei.

Als erstes lädt das Plugin die Listen bekannter Gegenstände und Talente aus dem Datenmodul. Dann wird die Klasse `ReadRound` instanziert, die die ganzen Informationen einliest. Für jede Region instanziert sie eine `ReadRegion`, die sich wiederum die Informationen aus der Datei liest. Gegenstände und Talente werden zuerst in der Liste der bekannten gesucht, falls sie noch nicht vorhanden sind, werden sie neu hinzugefügt.

[5] Leider bringen die Anführungszeichen nicht viel, weil einige Texte selber wiederum welche enthalten.

Kapitel 7

Bedienungsanleitung

Diese Anleitung ist bewusst kurz gehalten. Die meisten Operationen sollten intuitiv Verständlich sein. Der wichtigste Abschnitt ist die Konfiguration 7.4, weil dort einiges nicht so Eingängig gelöst ist.

Die Abbildungen sollen einen Eindruck des Programmes geben, wenn es nicht ausgeführt werden kann. Falls möglich sollte während der Lektüre dieses Kapitels VisEiA ausgeführt werden, um die beschriebenen Möglichkeiten gleich zu testen.

7.1 Installation

Die Installation wird von einem automatischen Installationsprogramm ausgeführt. Das System muss mindestens Windows 95 sein, die minimalen Hardwareanforderungen sind nicht getestet. Es wird keine andere Software benötigt, damit VisEiA funktioniert. Ohne Internetzugang kann allerdings kein Email-Spiel gespielt werden.[1]

Die Installation belegt rund 15 MB Plattenspeicher. Durch das Laden von Auswertungen wächst die Speicherbelegung der Datenbank an.

7.2 Erster Start

Beim ersten Start stellt VisEiA fest, dass noch kein Spiel registriert ist und stellt den Dialog für ein neues Spiel in den Vordergrund. Zudem sind noch keine Plugins registriert, weshalb der PluginBrowser ebenfalls erscheint. Mit diesem muss mindestens ein Plugin zum Einlesen von Auswertungen gewählt werden. (z.Z. gibt es nur den Parser für Eidolon, der als Datei `EidolonPars.dll` im Installationsverzeichnis liegt. Dieses muss hier ausgewählt werden.) Der PluginBrowser wird in Abschnitt 34 erklärt.

Im Dialog "Spiel erstellen" (Abbildung 7.1) wird dann ein Name für das Spiel angegeben und aus einer Liste das Plugin ausgewählt. Ein Plugin zum Erstellen neuer Züge gibt es leider noch nicht. Nun muss noch das Verzeichnis angegeben werden, in dem die Daten gespeichert werden sollen. Dieses Verzeichnis sollte unbedingt leer sein, da automatisch Dateien erstellt werden und Viseia nicht läuft, wenn eine davon verschwindet.

[1] Das vorhandene Plugin basiert auf Textdateien, so dass der Internetzugang auf einem anderen Computer sein kann.

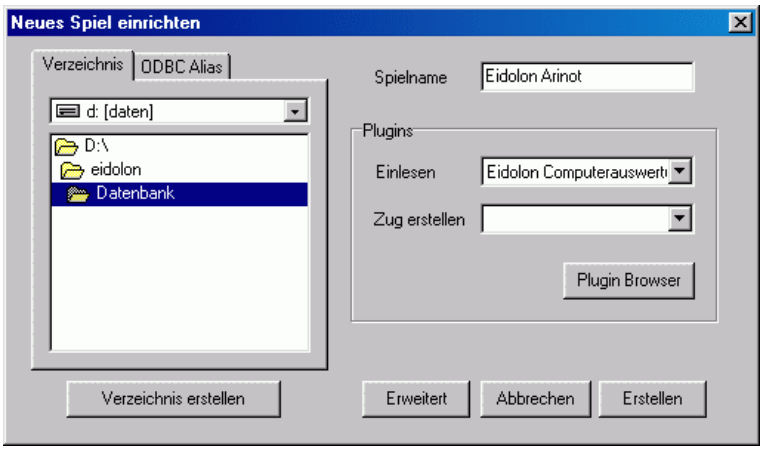
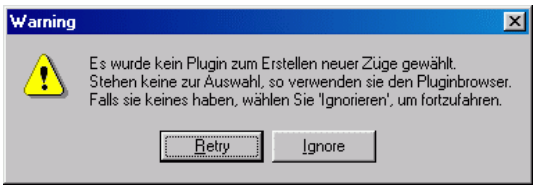


Abbildung 7.1: Dialog zum Einrichten eines neuen Spieles.

Für Experten gibt es die Möglichkeit, die SQL Anweisungen zu verändern. Nach einem Klick auf *Erweitert* kann der Pfad zu den Dateien mit den SQL Anweisungen geändert werden. Man kann nun die dort angegebene Datei als Vorlage nehmen und die SQL Statements anpassen.



Wenn man *Erstellen* wählt, erscheint die abgebildete Warnung, man habe kein Plugin zum Erstellen neuer Züge gewählt. Diese Warnung kann vorläufig nur mit Ignorieren entfernt werden, es gibt noch keine entsprechenden Plugins.

Als letzter Schritt kommt nun der Dialog zum Einlesen einer Auswertung, der in Abschnitt 7.3.4 beschrieben wird. Wurde die Auswertung erfolgreich gelesen, erscheint nun das Hauptfenster.

7.3 Grundfunktionen

Im Hauptfenster gibt es links ein Register von Steuerflächen (Siehe Abbildung 7.2). Man kann zwischen Übersicht, Statistik und Suche wählen. Beim Ausführen von Abfragen wird das Ergebnis auf der rechten Seite nach vorne gebracht. Zudem gibt es im rechten Bereich eine Seite mit den Meldungen, die zu dieser Runde gehören.

7.3.1 Karte betrachten

Die Karte in Abbildung 7.2 kann mit den Lupensymbolen vergrössert und verkleinert werden. Je grösser die einzelnen Felder sind, desto mehr Daten können eingeblendet werden. In Abschnitt 7.4 ist beschrieben, wie man Gegenstände zum Einblenden auswählt. Die ausgewählten Gegenstände erscheinen als Icon rechts oben. Klickt man sie an, werden sie vertieft dargestellt und die Anzahl für jede Region eingeblendet. Durch nochmaliges anklicken werden sie wieder ausgeblendet.

Auf der linken Seite ist ein Baum, in dem weitere Informationen zu den Regionen sind. Durch Anwählen einer Region wird die Karte auf diese zentriert. Wenn man auf das '+'

neben der Region klickt, kann man sich weitere Informationen anzeigen lassen. In diesem Baum werden alle Regionen angezeigt. Wenn zur Zeit keine Einheit in der Region ist, werden die Daten der am wenigsten alten Auswertung verwendet, um Informationen wie Baumbestand oder Wildpferde anzuzeigen.

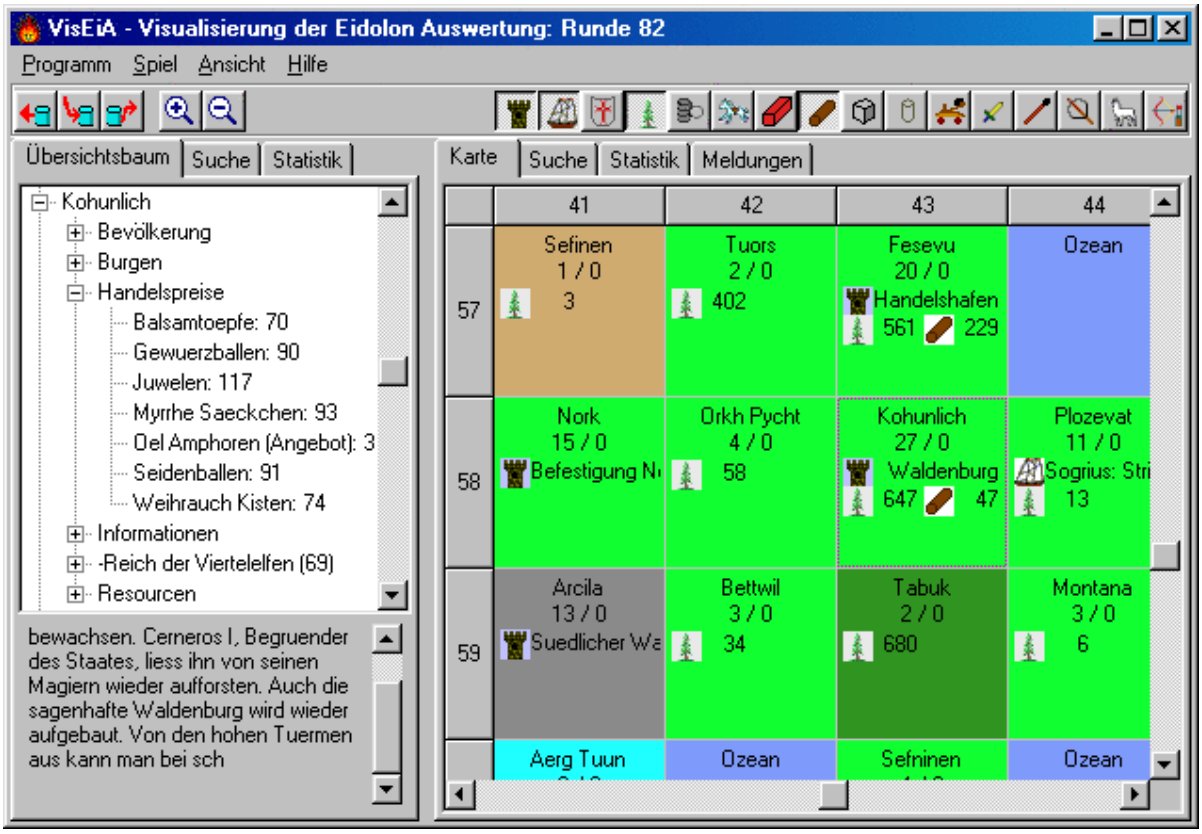


Abbildung 7.2: Das Hauptfenster mit der Karte und eingeblendeten Informationen.

7.3.2 Statistik erstellen

Statistiken können über Gegenstände oder Talente erstellt werden, mit dem Formular aus Abbildung 7.3.

Bei den Gegenständen kann man die Entwicklung über mehrere Runden betrachten. Es können mehrere Gegenstände ausgewählt werden, zum Beispiel alle Handelsgüter. Die Gegenstände werden summiert, ebenso wie Gegenstände der Einheiten in der Region summiert werden. Wählt man “Regionen einzeln”, so wird für jede gewählte Region eine eigene Kurve dargestellt, ansonsten wird über alle summiert.

Wählt man eine andere als die eigene Partei, muss man sich bewusst sein, dass nur das angezeigt wird, was die eigenen Einheiten erkennen können. Mehr Informationen sind in der Auswertung nicht enthalten.

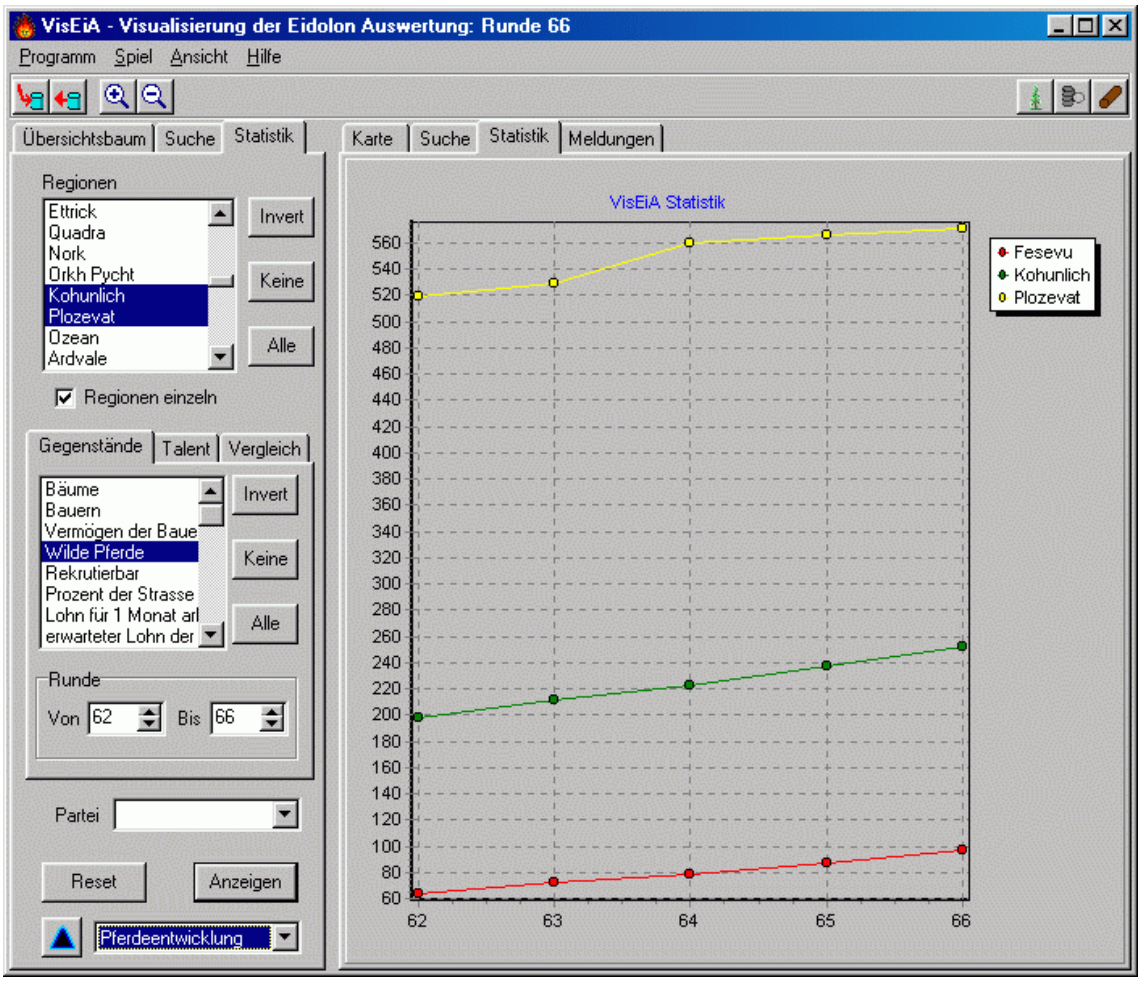
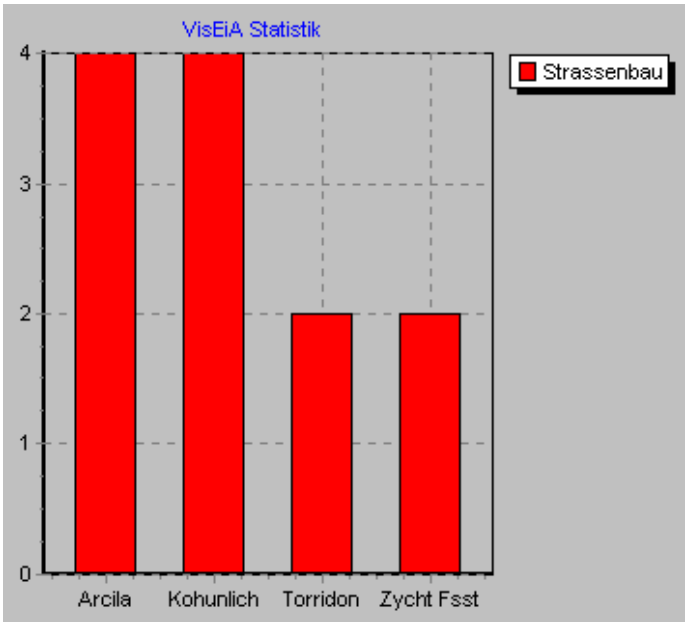
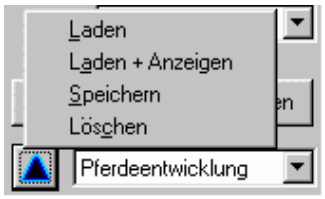


Abbildung 7.3: Mit den Statistiken lässt sich die längerfristige Entwicklung betrachten und vergleichen.



Für Talente kann nur ein Vergleich zwischen Regionen gemacht werden, da diese nur für die aktuelle Runde bekannt sind. Dargestellt wird jeweils die maximale Talentstufe in einer Region (siehe nebenstehende Abbildung). Regionen, in denen das Talent nicht vorkommt, werden ausgeblendet. Dadurch kann man einen Überblick über das "Bildungsniveau" der Einheiten in verschiedenen Regionen gewinnen.



Statistiken kann man speichern, erneut laden und eine gespeicherte Konfiguration löschen. Durch einen Mausklick auf den blauen Pfeil erscheint das nebenstehend abgebildete Menü. Welche Statistik neu geladen werden soll, wird in der Liste neben dem blauen Schalter bestimmt.

Zu beachten ist, dass der Zeitraum der Statistik mit gespeichert wird. Bei geladenen Angaben muss man eventuell noch die Runden ändern, damit auch die neusten Daten berücksichtigt werden.

7.3.3 Einheiten / Gegenstände suchen

Die Suche (Abbildung 7.4) erlaubt es, nach Einheiten zu suchen, die bestimmte Talente oder Gegenstände haben. Man kann auch Gegenstände suchen, die direkt in den Regionen sind; das Ergebnis einer solchen Suche sind alle Einheiten, die sich in einer entsprechenden Region befinden. Es müssen nicht alle Felder ausgefüllt werden, meist werden nur wenige Felder gefüllt.

Die Fragen lassen sich durch ein logisches "und" oder ein "oder" verknüpfen. Ein Block mit "und" muss durchgehend erfüllt sein, um gefunden zu werden. Nach einem "oder" beginnt ein neuer Block. Abfragen der Art `(A oder B) und C` lassen sich nicht erstellen.

Das Ergebnis wird als Tabelle dargestellt. Wenn man auf eine Zeile klickt, springt man in der Karte auf die entsprechende Region.

Leider ist das Abspeichern von Suchabfragen nicht implementiert.

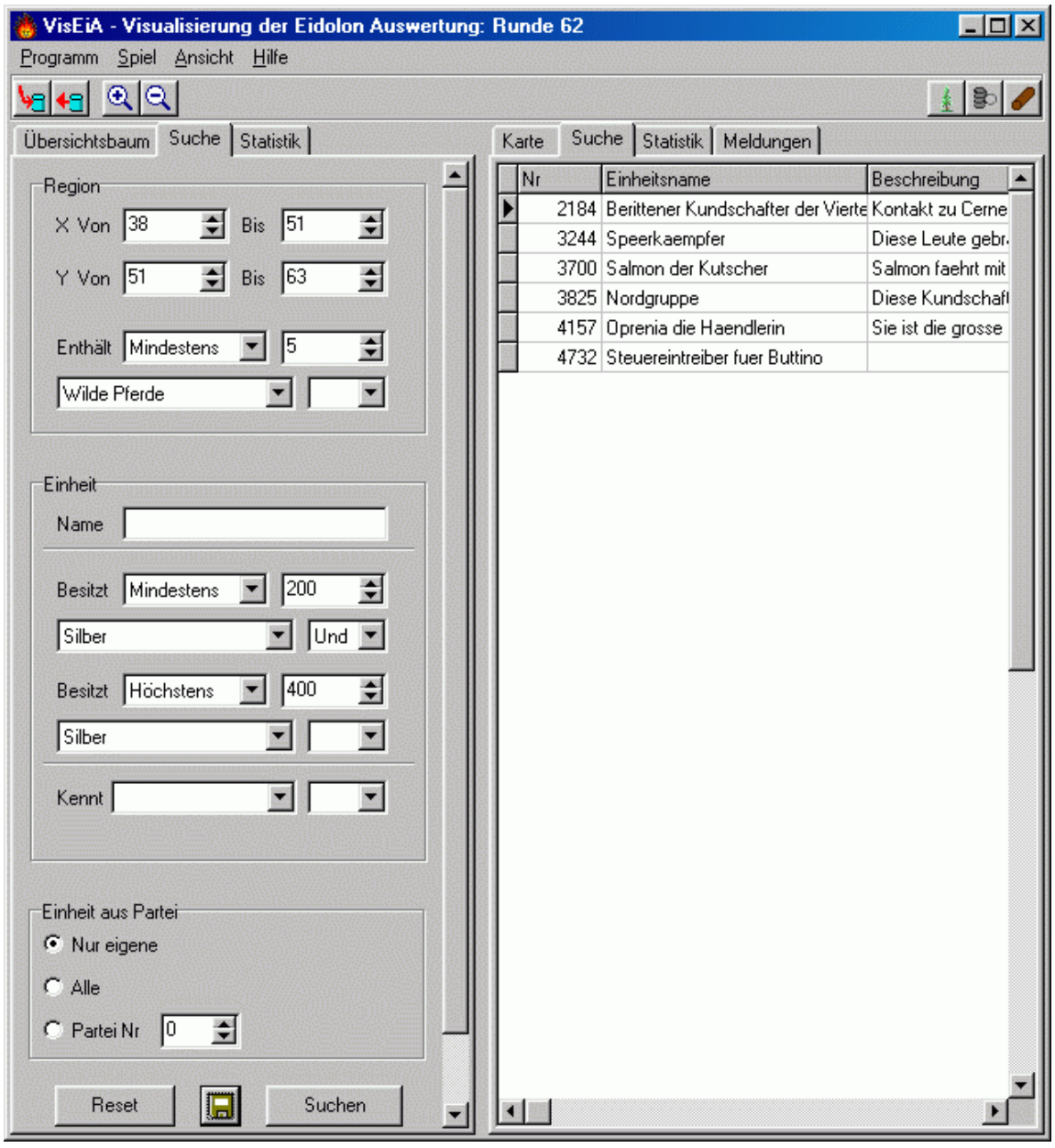
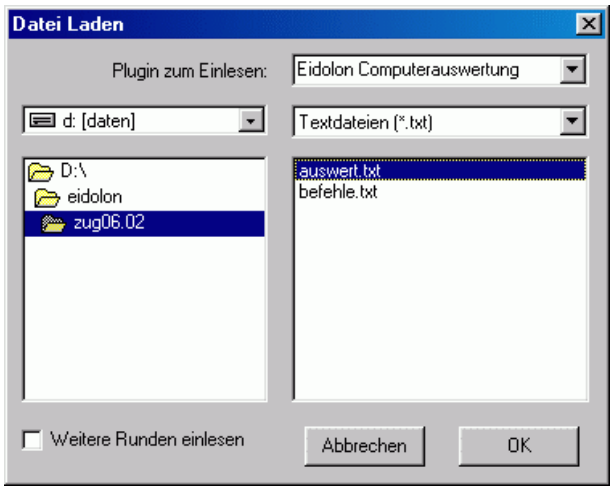


Abbildung 7.4: Mit der Suche kann man nach Talenten oder Gegenständen suchen.

7.3.4 Auswertung einlesen



Um VisEiA mit Daten zu füttern, müssen Auswertungen eingelesen werden. Der Dialog in nebenstehender Abbildung zum Auswählen der Auswertungs - Datei kann beim Start von VisEiA aufgerufen werden, oder vom Hauptfenster aus unter "Spiel, Auswertung einlesen".

Es wird die Datei mit der Auswertung angegeben. Rechts oben kann zudem das Plugin ausgewählt werden. Standardmässig ist dasjenige gewählt, das beim Erstellen des Spiels angegeben wurde.

Wählt man das Feld "Weitere Runden einlesen", so werden die eingelesenen Daten noch nicht dargestellt, sondern dieser Dialog erscheint nochmals. So kann man nacheinander mehrere Runden einlesen, ohne viel Zeit durch die Darstellung auf der Karte zu verlieren.

7.4 Konfiguration

Es gibt zwei wichtige Dinge, die mit dem Dialog in Abbildung 7.5 konfiguriert werden können.

Einmal kann man einstellen, welche Gegenstände auf der Karte eingeblendet werden können. Dazu wählt man Einträge der Liste rechts oben, die dann in der Liste links daneben erscheinen. Aus dieser Liste können sie wieder entfernt werden. Es können maximal 18 gewählt werden. Burg und Schiffe sind immer in der Liste der anzeigbaren Gegenstände.

Zum anderen können die Daten vieler Spielobjekte angepasst werden, jeweils soweit dies nicht im Spiel möglich ist.[2] Insbesondere kann für fast alles ein Icon vergeben werden. Man kann auch eigene Bilder[3] hinzuladen. Burgen und Schiffe können ein spezielles Icon erhalten, und jeder Gegenstandstyp hat sein Icon.

Als weitere Optionen kann man einstellen, ob alte Daten gelöscht werden sollen. Wenn eine Zahl grösser 0 angegeben wird, so löscht VisEiA beim Laden einer neuen Auswertung alle Daten, die um mehr als diese Zahl älter sind.

Die Debug Stufe entscheidet, wie viele Informationen in die Datei `debug.txt` im VisEiA Verzeichnis geschrieben werden.

[2]Es würde keinen Sinn machen, die Beschreibung einer Burg zu ändern, da diese beim Einlesen der nächsten Auswertung überschrieben wird. Eine Burg beschreibt man in den Befehlen. Einen Gegenstand hingegen kann man im Spiel nicht beschreiben, deshalb kann er hier beschrieben werden.

[3]Im .bmp Format, 16 x 16 Pixel

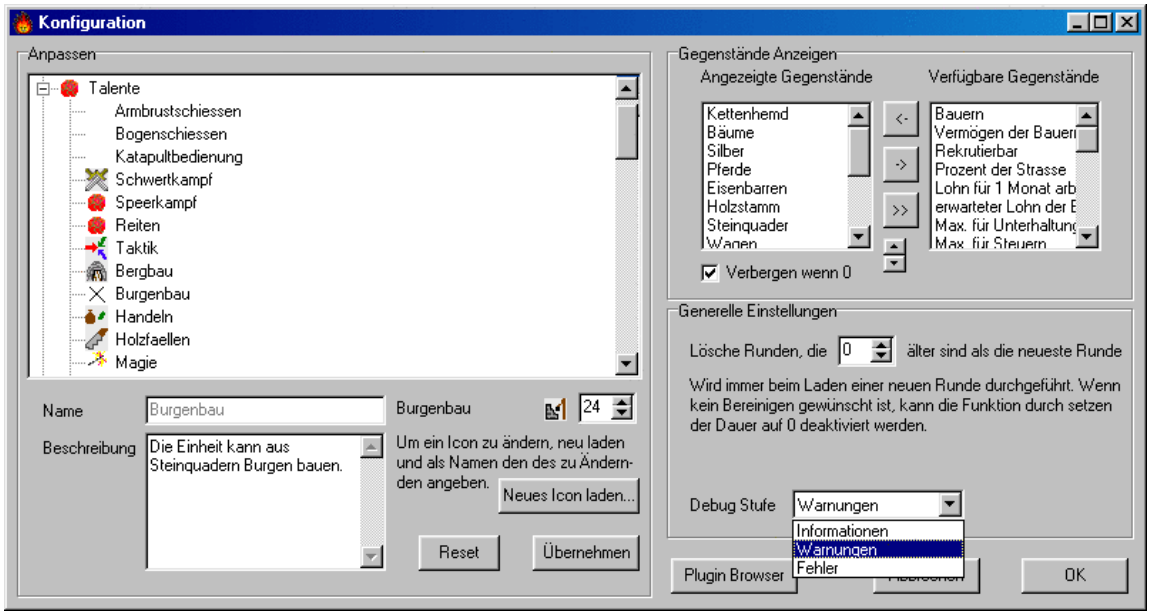


Abbildung 7.5: Der Konfigurationsdialog erlaubt es, VisEiA an individuelle Bedürfnisse anzupassen.

Plugin Browser

Der Plugin Browser (Abbildung 7.6) erlaubt das Hinzufügen neuer Programmteile zum Einlesen von Daten oder zum Erstellen von Zügen. Wenn man ein Plugin hinzugefügt, so wird es in ein Unterverzeichnis von VisEiA kopiert, wo es später wieder gefunden wird. Beim Entfernen wird die Datei nicht gelöscht, sondern lediglich die Informationen aus der Registry entfernt.

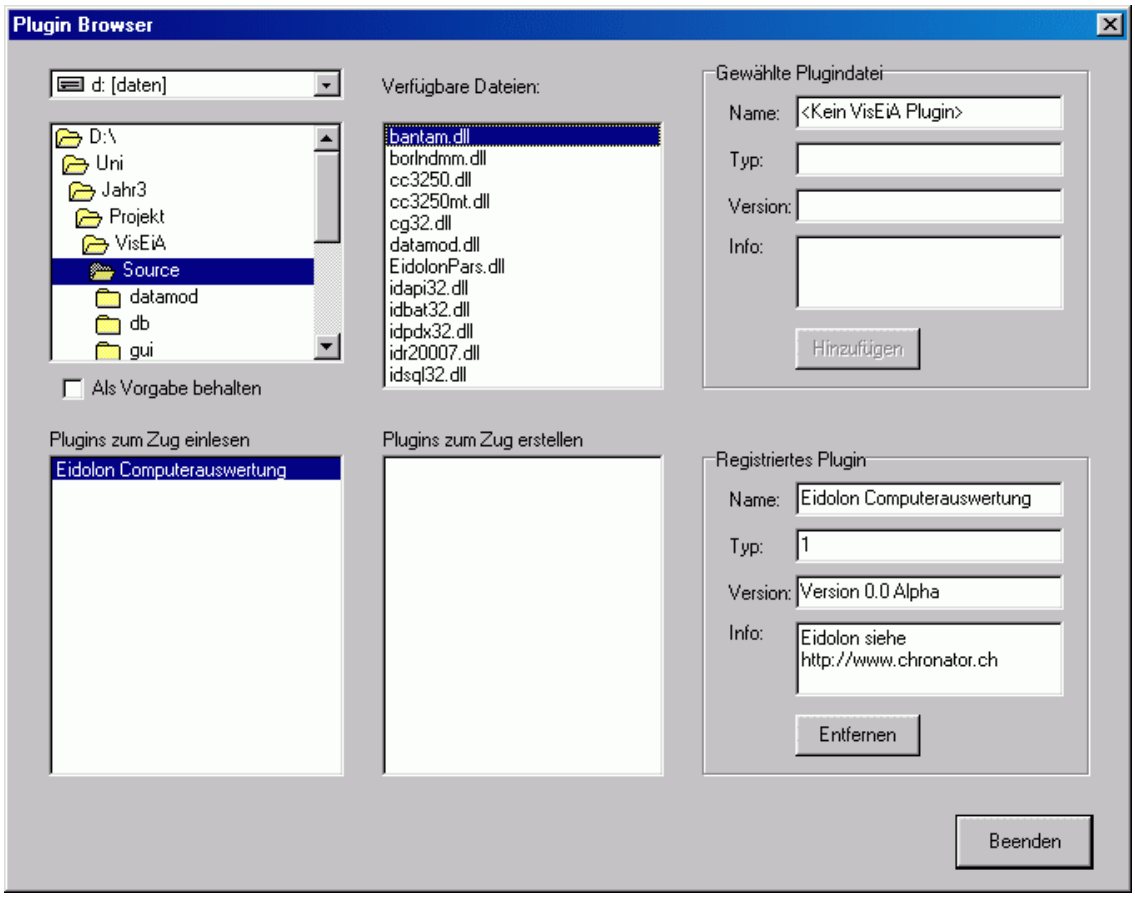


Abbildung 7.6: Mit dem Plugin Browser werden neue Plugins hinzugeladen und alte entfernt.

Kapitel 8

Zusammenfassung und Ausblick

Das letzte Kaptitel dient einem kritischen Rückblick auf das Erreichte und zeigt Möglichkeiten zur Weiterentwicklung.

8.1 Erreichte Ziele

Die Ziele wurden im grossen und ganzen erfüllt. Der Vergleich zwischen verfügbaren Technologien hat zu einer Wahl geführt, die sich als durchfürbar erwies.

Das Programm stellt Informationen geographisch dar, man kann sich Statistiken anzeigen lassen und die Daten durchsuchen. Die Speicherung in einer Datenbank funktioniert und ermöglicht die umfangreichen Such- und Statistikmöglichkeiten.

Die formalen Kriterien wurden ebenfalls einigermassen erfüllt. Die Installation verläuft weitgehend automatisch und es wird kein Datenbankserver gebraucht. Die Stabilität des Programmes lässt leider zu wünschen übrig. Wenn nicht alles wie erwartet läuft, kann es zu Abstürzen kommen. Die Benutzung scheint dem Autor ausreichend intuitiv, doch muss das mit Anwendern getestet werden. Die Flexibilität wurde mit den Plugins gut gelöst. Die Datenbank könnte mehr Reserven für zusätzliche Informationen bieten, aber mit den Methoden des Datenmoduls für direkte SQL Statements ist zumindest eine Lösung angeboten.

8.2 Persönliche Einschätzung

Der Arbeitsaufwand, um ein komplettes Programm zu schreiben, wurde eindeutig unterschätzt. Gegen Ende der Arbeit mussten deshalb Abstriche gemacht werden und nur noch das Nötigste implementiert werden. Es ist aber gelungen, ein funktionierendes Programm zu schaffen.

Die Arbeit mit CBuilder hätte in zwei Punkten wesentlich vereinfacht werden können.

Der ganze Aufwand für das Datenmodul wäre wesentlich kleiner gewesen, wenn eine objektiorientierte Datenbank verwendet worden wäre. Die Objekte hätten sich mehr oder weniger von selber in die Datenbank geschrieben und sich auch selbst gelesen. Reines serielles Abspeichern[1] wäre nicht ausreichend gewesen für Statistiken und Suchen, da

[1] Im Sinne von Java's `serialize`

diese Funktionalität sonst nicht von einer Datenbankmaschiene geboten worden wäre.

Der Anspruch, die Schnittstellen in ANSI C++ zu halten, hat den Aufwand auch wesentlich erhöht (zudem konnte er nicht konsequent eingehalten werden). Lapalien wie das pausenlose konvertieren zwischen C++ `string` und dem CBuilder `AnsiString`[2] frassen viel Zeit, die nützlicher hätte verwendet werden können.

Hätte der Autor zu Beginn der Arbeit schon die Vorlesung "Documents Multimedia"[5] abgeschlossen, wäre der Ansatz mit XML besser abgeschnitten. Mit der Umwandlungs- und Abfragesprache XSL steht ein Werkzeug zur Verfügung, um XML Daten automatisch zu bearbeiten. Ob dies einfacher oder schwieriger geworden wäre als die Arbeit mit selbstgeschriebenen Klassen und SQL, kann nicht beurteilt werden. Doch ohne das Wissen aus der Vorlesung war XSL keine Möglichkeit.

Einige Ideen zur Informationsdarstellung gehen in Richtung Data Mining. Da dieses Thema dem Autor erst gegen Ende der Arbeit bekannt wurde, konnte nicht mehr genau auf diese Konzepte eingegangen werden.

8.3 Probleme in der Programmierung

Während der Programmierung tauchten einige spezielle Fehler auf. Ein Teil davon waren Programmierfehler, andere aber Bugs von verwendeten Klassen. Drei sollen hier vorgestellt werden.

Eine Besonderheit der Arbeit mit DLL's ist der Speicherbereich. Die DLL hat einen eigenen Heap Bereich, von dem mit `new` Speicher alloziert wird. Speicher, der in der DLL alloziert wurde, muss auch in dieser wieder freigegeben werden. Für CBuilder schafft eine spezielle Bibliothek Namens `memmgr.dll` Abhilfe, sie verwaltet den ganzen Programmspeicher anders.

Vermutlich ein Bug ist, dass die Blob Felder einer Datenbank nicht über die Felder eines Objekts vom Typ `TQuery` geschrieben werden können, sondern nur mit einer `TTable`. Ansonsten wurde immer die TQuery verwendet, die ein SQL Statement kapselt. Zum Schreiben eines Blobs wird eine TTable verwendet, die eine ganze Tabelle kapselt. Die richtige Zeile wird über einen Filter gewählt.

Ein eigenartiger Fehler trat mit dem `TBDChart` auf. Dieser Diagrammklasse kann als Datenquelle eine `TQuery` zugewiesen werden, was in unserem Fall sehr praktisch ist. Doch wenn die TQuery im Datenmodul erzeugt wurde, also nicht auf dem selben Heap, gab es einen Speicherzugriffsfehler. Als Notlösung wird die TQuery nun in VisEiA erzeugt und dem Datenmodul als Pointer übergeben. So funktionert alles tadellos.

8.4 Mögliche Verbesserungen und Erweiterungen

VisEiA hat recht lange Ladezeiten. Wenn eine ganze Runde aus der Datenbank gelesen wird, kann das durchaus eine halbe Minute dauern. Während dieser Zeit wird die Darstellung nicht aktualisiert und das Programm reagiert nicht. Eine minimale Indikation wurde

[2] Obwohl er AnsiString heisst, ist dieser Typ nicht ANSI C++, sondern wurde wegen der Kompatibilität zu Pascal in CBuilder aufgenommen. Weil die VCL in Pascal geschrieben ist, brauchen alle VCL Klassen AnsiString.

implementiert: der Mauszeiger verwandelt sich in eine Sanduhr. Aber bei dieser Dauer wäre es besser, wenn irgendwo ein Streifen gezeigt würde, der den Fortschritt anzeigt. Eventuell wäre zudem eine Möglichkeit vorzusehen um den Ladevorgang abbrechen zu können. Noch besser wäre natürlich eine Optimierung der Ladezeiten.

Die Statistikerzeugung ist nicht vollständig. Interessant wäre insbesondere ein Vergleich der Entwicklung zweier Parteien. Auch eine Möglichkeit zum Drucken von Statistiken wäre nicht schlecht. Die verwendete Komponente `TDBGrid` bietet Funktionalität zum Drucken an, so dass diese Aufgabe relativ einfach sein sollte.

Eine wichtige Erweiterung wäre ein Plugin zum Erstellen neuer Züge. Um Bewegungen auf der Karte zu befehlen, könnte es `TMapDrawGrid` verwenden. Das Plugin sollte sicher stellen, dass nur gültige Befehle gegeben werden und dass die Entwicklung berücksichtigt wird. Dadurch könnte es z.B. warnen, wenn Ressourcen nicht ausreichen werden.

Ein Plugin könnte zum Beispiel auch keine Befehle erstellen, sondern beim Aufruf die Karte als HTML exportieren für die Webseite des Spielers oder Daten für ein anderes Programm. Weiter besteht die Möglichkeit, Plugins für andere Spiele zu erstellen.

Nicht zuletzt sollten auch die Fehler korrigiert werden, die jetzt schon bekannt sind. Um die Arbeit termingerecht fertigzustellen, wurden sie nicht mehr korrigiert, sondern nur noch notiert. Die Datei `bugs.txt` im Programmverzeichnis listet alle bekannten Fehler auf.

Anhang A

Glossar

A.1 Datenbank

DBMS **D**ata**b**ase **M**anagement **S**ystem. Programmpacket, das das Erstellen, Betreiben und Verwalten einer Datenbank erlaubt.

SQL **S**tructured **Q**uery **L**anguage. Standardisierte Abfragesprache für relationale Datenbanken.

ODBC **O**pen **D**ata**b**ase **C**onnectivity. Ursprünglich von Microsoft entwickelte, später aber standardisierte Form zum Zugriff auf beliebige Datenbanken.

BDE **B**orland **D**atabase **E**ngine. Ergänzung zu ODBC, den Borland mit seinen Entwicklungsumgebungen mitliefert. Objektorientiert.

IBX **I**nter**b**ase E**x**press. Sammlung von CBuilder Klassen, um die API von Interbase direkt zu verwenden.

JDBC **J**ava **D**ata**b**ase **C**onnectivity. Analog zu ODBC, aber nicht auf Systemebene, sondern als Java Klassen implementierte Datenbanktreiber.

A.2 Extensible Markup Language

XML E**x**tensible **M**arkup **L**anguage. Standartisiertes Hierarchisches Dokumentformat

XSL e**X**tended **S**tylesheet **L**anguage . Verarbeitungssprache um XML Dokumente umzuformatieren.

DTD **D**ocument **T**ype **D**efinition. Definition der Struktur für ein XML Dokument.

SAX **S**imple **A**PI for **X**ML Processing. Konzept zur sequentiellen Verarbeitung von XML.

DOM **D**ocument **O**bject **M**odel. Konzept zur Verarbeitung von XML in Programmiersprachen, objektorientierte Baumstruktur

A.3 Produkte

CBuilder Entwicklungsumgebung für C/C++, mit RAD Bibliotheken. Produkt der Firma Borland/Inprise

Delphi Objektorientierte Weiterentwicklung von Pascal, bekannt für die Einführung von RAD. Produkt der Firma Borland/Inprise

Java Programmiersprache, die zu plattformunabhängigen Programmen kompiliert wird. Von einem Interpreter, der Java Virtual Machine, ausgeführt. Entwickelt von SUN, aber ein offener Standard.

Interbase Datenbanksystem mit Server - Client Architektur. Produkt der Firma Borland / Inprise

Access Einfaches Datenbankprogramm, dass mehr wegen dem graphischen Interface bekannt ist. Kein Datenbankserver und nur für Einzelbenutzer geeignet. Produkt von Microsoft

A.4 andere Begriffe

Portabilität Die Möglichkeit, Programme auf verschiedenen Systemen verwenden zu können. Entweder kann der Quellcode in verschiedenen Umgebungen kompiliert werden, oder das kompilierte Programm selber läuft in verschiedenen Umgebungen[1].

JFC **J**ava **F**oundation **C**lasses, Sammlung von Packages, die in jedem Java System verfügbar sein sollten.

RAD **R**apid **A**pplication **D**evelopment. Ein Umgebung, mit der man ohne grossen Aufwand ein Programm entwickeln kann. Nachteil sind uneffiziente Programme und grosse Installationsdateien.

API **A**pplication **P**rogramming **I**nterface. Bibliothek mit Funktionen, die aus anderen Programmen aufgerufen werden können, um die Funktionalität des Programmes im anderen Programm zu benutzen.

VCL **V**isual **C**omponents **L**ibrary. Objektbibliothek, die den Kern von Delphi / CBuilder ausmacht.

[1] Java ist die wichtigste Sprache, bei der kompilierte Programme auf mehreren Systemen laufen.

Literaturverzeichnis

[1] Flanagan, David: Java in a Nutshell. O'Reilly, Sebastopol (USA), 3. Auflage, 1999

[2] Lafore, Robert: Object-Oriented Programming in C++. SAMS Publishing, Indianapolis (USA), 3. Auflage, 1998

[3] Louden, Kenneth C.: Programming Languages: Principles and Practice. PWS, Boston (USA), 1993

[4] Meier, Andreas: Relationale Datenbanken. 3. Auflage, Heidelberg 1998

[5] Vanoirbeek, Christine und Abu Chaled, Omar: Documents multimédia - XML. Vorlesung WS 2000, Fribourg (CH), http.//mis.eif.ch (1.11.2000)

[6] Borland Software Corporation `http://www.borland.com` [2]

[7] Borland Newsgroups:
news://borland.public.cppbuilder.database.desktop
news://borland.public.cppbuilder.graphics
news://borland.public.cppbuilder.database.interbaseexpress
news://borland.public.cppbuilder.vcl.components.writing
news://borland.public.bde

[8] Sun Corporation: Java. `http://java.sun.com`

[9] Sun Corporation: Java 1.3, Dokumentation, `http://java.sun.com/j2se/1.3/docs`

[10] jGuru.com: JDBC 2.0 Fundamentals, Short Course.
`http://developer.java.sun.com/developer/onlineTraining/Database/JDBC20Intro/JDBC20.html`

[11] Sun Corporation: `http://java.sun.com/xml`

[12] Apache Group: `http://xml.apache.org`

[13] Bosak, Jon und Bray, Tim; XML and the Second-Generation Web.
`http://www.sciam.com/1999/0599issue/0599bosak.html`

[14] Borland InterBase: `http://www.interbase.com`

[15] Extendes Systems: Advantage Database Server.
`http://www.advantagedatabase.com/ADS/default.htm`

[2] Die URL's wurden soweit nicht anders angegeben zuletzt am 7.9.2001 besucht.

[16] MySQL: http://www.mysql.com (15.6.2001)

[17] Atlantis project web page. http://www.prankster.com/project

[18] German Atlantis Homepage.
Offiziel (z.Z. defekter Link) http://home.pages.de/$\sim$GA
Gefunden unter: http://www.geocities.com/kensanata/atlantis

[19] Chronator Spiele Magazin: http://www.chronator.ch

[20] Meng, Roman: Eidolon - Eine Atlantis Welt. Version 1.02,
http://www.chronator.ch/spiele/eidolon/index.html

[21] EidolonC, Autor nicht feststellbar.
ftp://ftp.phantasia.org/chronator/tools/eidoloncs12.zip

[22] Frank Schnekenbuehl: AChart.
http://www.geocities.com/SouthBeach/Marina/1645/achart.html

[23] Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307,
USA. http://www.gnu.org

[24] WhatIs: http://whatis.techtarget.com/