Protokollierung des Spielerverhaltens in Massive Multiplayer Online Games

Bachelorarbeit
von

Michael Willigens

aus
Düsseldorf

vorgelegt am

Lehrstuhl für Rechnernetze
Prof. Dr. Martin Mauve
Universität Düsseldorf

August 2005

Korreferent:
Prof. Dr. Stefan Conrad

Betreuer:
Prof. Dr. Martin Mauve

Anmerkungen

Folgenden Personen gilt mein Dank, ohne ihre Unterstützung wäre diese Arbeit nicht möglich gewesen:

Prof. Dr. Martin Mauve, für den Vorschlag und die Betreuung dieser Arbeit.
Meiner Familie und meinen Freunden für geistige Unterstützung während dieser Zeit.
Julian Swagemakers und Sebastian Findler für das Testen während der Programmierung.
Kai M. Krähahn und Christoph Pälmer, für einige kreative Ideen.
Rustak, dem Erschaffer der World of Warcraft Wikipedia.
Der Firma Blizzard, weil sie so ein großartiges Spiel entwickelt hat.

Inhaltsverzeichnis

Abbildungsverzeichnis

Glossar

AddOn	Interfacemodifikation/-erweiterung
API	Application Program Interface - Anwendungs-Programmierschnittstelle
Event	Ereignis/Spielereignis
Genre	Spielkonzept bzw. Spielprinzip
Instanz	Eine kleine abgeschlossene Spielwelt. Dient der Erledigung von umfangreichen Quests mit einer Gruppe.
Interface	Benutzerschnittstelle
LUA	Eine Programmerweiterungs-Scriptsprache
LUA – VM	LUA Virtual Mashine
MMO	Massive(ly) Multiplayer Online
MMOG	Massive(ly) Multiplayer Online Game
MMORPG	Massive(ly) Multiplayer Online RolePlaying Game
MOG	Multiplayer Online Game

NPC	Non Player Character - Ein computergesteuerter Spielcharakter
Party	Eine Spielergruppe von bis zu fünf Spielern
PvE	Player versus Environment - Spieler gegen Monster/NPC
PvP	Player versus Player - Spieler gegen Spieler
PvX	Player versus X - Spieler gegen Spieler/NPC/Montster
Quest	Ein Abenteuer in einem Rollenspiel
Raid	Ein Schlachtzug von bis zu 40 Spielern
Realm	Die Bezeichnung einer virtuellen Spielwelt
TOC	Table of Contents - In WoW werden TOC Dateien benötig, um Add-Ons einzubinden.
WoW	World of Warcraft

Kapitel 1

Einleitung

Massive Multiplayer Online Games (MMOGs) sind Computerspiele, bei denen mehrere tausend Teilnehmer gleichzeitig in einer Spielwelt interagieren können. Aufgrund ihrer Komplexität sind die meisten MMOGs mit monatlichen Kosten verbunden. Derzeit zahlen fünf Millionen Spieler monatliche Gebühren für die Teilnahme an MMOGs (Quelle: MMOGchart [3]). Ein Spielkonzept dieses Zweiges sind MMORPGs. Bei diesen Spielen sind die Teilnehmer an einem Rollenspiel beteiligt. Dabei sieht jeder die Welt aus der Sicht seines Spielcharakters. Verbunden mit vielfältigen Interaktionsmöglichkeiten (Chat, Handel, Gruppenbildungen, virtuelle Kriege, etc.) entstehen hierdurch interessante Parallelen zu der realen Welt. MMORPG Spielwelten unterliegen beispielsweise wirtschaftlichen Effekten wie Inflation und Deflation der virtuellen Zahlungsmittel. Das liegt unter anderem daran, dass die virtuellen mit den realen Wirtschaftssystemen verknüpft sind. Zum Beispiel wird in Onlineauktionshäusern reger Handel mit den Gegenständen, den virtuellen Zahlungsmitteln und sogar den Charakteren selbst getrieben. Um das Spiel für den Anwender interessant zu halten, müssen sich die Entwickler Gedanken darüber machen, wie sie zu starke Schwankungen der virtuellen Wirtschaft verhindern können.
Aus soziologischer Sicht sind Effekte wie Gruppenbildungen, virtuelle Freund- und Feindschaften und Geschlechterverhalten durchaus näherer Untersuchung wert. Psychologische Untersuchungen darüber, wann und warum ein Teilnehmer viel oder wenig Zeit mit dem Spiel verbringt oder welchen virtuellen Tätigkeiten er in dieser Zeit nachgeht, wären interessant.

Aufgrund ihrer Verbreitung stellen MMORPGs ein neues Massenmedium dar. Welche

Einflüsse MMORPGs tatsächlich auf ihre Teilnehmer ausüben wurde noch viel zu ungenügend untersucht. Das Problem, welches sich hier stellt, ist die Frage wie man zuverlässig Informationen sammelt, die als Grundlage jeder Untersuchung nötig sind.

1.1 Aufgabenstellung

Die Aufgabenstellung dieser Arbeit ist es, eine Basis zu entwickeln, mit der das Spielerverhalten in einem MMORPG automatisiert aufgezeichnet werden kann, damit der Benutzer keine Fragenbögen irgendeiner Art ausfüllen muss. Es handelt sich hierbei um den Titel “World of Warcraft“ der Firma “Blizzard Entertainment“ für den ein Interface Plugin entwickelt werden soll, das eine automatisierte Aufzeichnung der dynamischen Spieldaten (Quests, Aufenthaltsort, Charakterdaten, Gruppenmitglieder, Gegner etc.) ermöglichen soll. Der Zweck ist es, dieses Programm bei möglichst vielen Spielern einzusetzen um eine spätere Auswertung der gesammelten Daten zu erlauben.

Kapitel 2

Referenzprojekte

In diesem Kapitel werden vorhandene Arbeiten und Studien, die es auf dem Gebiet des Sammelns und Analysierens von MMORPG-Daten gibt, vorgestellt.

2.1 The Daedalus Project

Nick Yee gründete 2003 “The Daedalus Project“ [15]. Basierend auf seinen Daten, die er seit 1999 sammelt, veröffentlicht er hier seine Studien über die psychologischen Effekte von MMORPGs. Bis heute hat er mehr als 35.000 Online-Fragebögen ausgewertet. Mit dieser Datenmenge ist er in der Lage, repräsentative Schlussfolgerungen über das Verhalten der Spieler zu machen. Folgende Aspekte werden von ihm untersucht: Konsumverhalten, Geschlechter- und Altersverteilung, Rollenverhalten, virtuelle Freundschaften, Gruppenverhalten sowie Abhängigkeit vom Spiel. Er veröffentlichte seit Start seines Projektes mehr als 80 Artikel, die nach Themen geordnet über das "Daedalus Gateway"[14] abrufbar sind.
Der Unterschied zu dieser Arbeit besteht darin, dass das Verhalten der Testpersonen nun automatisiert und nicht per Fragebogen aufgezeichnet werden soll. Dies unterbindet eine möglicherweise subjektive Verfälschung der Daten. Das ist besonders bei der Untersuchung des Konsumverhaltens von Wert. Wer wird offen zugeben, wie viel Zeit er tatsächlich in ein Computerspiel investiert?

2.2 Thottbot

Thottbot [13] ist eine umfangreiche Webdatenbank, deren Sinn es ist alle relevanten Spielinformationen der "World of Warcraft"Welt vorzuhalten und transparent darzustellen. Als Grundlage hierzu dient ebenfalls ein Interface AddOn, welches die Daten während einer Spielsitzung sammelt. Das Hauptaugenmerk liegt hierbei nicht auf einem zeitlich genauen Spielablaufsprotokoll, sondern auf dem Sammeln von statischen Spieldaten. Thottbot sammelt beispielsweise die Positionen von NPCs oder die Wahrscheinlichkeiten mit der gewisse Gegenstände von Monstern erbeutet werden können. Das Thottbot Project stellt einen Uploader zur Verfügung, mit dem die gesammelten Informationen dann automatisch vor Spielbeginn zu der Webdatenbank hochgeladen werden.

2.3 WarcraftRealms

Bei WarcraftRealms [5] handelt es sich, wie bei Thottbot [13], um ein durch ein AddOn gestütztes Projekt. Hierbei wird die Bevölkerungsdichte sowie die Klassen-, Rassen-, und Levelverteilung auf den World of Warcraft Servern untersucht. Diese Daten geben auch Aufschluss über die Verteilungen in den Gilden, sowie deren Platzierungen untereinander. Das AddOn, mit dem diese Daten gesammelt werden, heißt “Census“. Derzeit basieren die Informationen auf den gesammelten Daten von 12 Millionen Spielcharakteren.

2.4 MMOG Chart

MMOG Chart [3] ist ein freies Projekt, welches die Anzahl der Abonnementen von MMOGs veröffentlicht. Beispielsweise werden hier auch die zeitlichen Veränderungen der Abonnementen festgehalten und in geeigneten Diagrammen dargestellt. Einschneidende Ereignisse wie z.B. die Veröffentlichung von Titeln wie “Everquest“ oder “World of Warcraft“ spiegeln sich hier wieder. Die Informationen, auf denen dieses Projekt basiert, werden jedoch von sehr verschiedenen Quellen zusammengetragen.

Presseveröffentlichungen, Nachrichtenartikel, Entwicklerforen, anonyme Quellen usw. dienen als Grundlage. Es ist leider nicht möglich die Zahl der Abonnementen genauer zu Erfassen, da die Entwickler meist keinen freien Einblick in ihre Daten zulassen. Zudem kann dieses Problem auch nicht statistisch untersucht werden, da es nicht möglich ist sinnvolle Hochrechnungen auf die Benutzerzahlen zu machen.

2.5 Aktuelles

Computerspiele gewinnen auf Grund ihrer Verbreitung immer mehr an Medienpräsenz. Wie ein aktueller Artikel aus dem “Spiegel“ [9] beschreibt, wachsen Computerspiele von einer Randerscheinung zu einem Volkssport heran. Es werden immer neue wirtschaftliche Rekordmarken auf diesem Gebiet gesetzt. Beispielsweise geht Jörg Trouvain, der Geschäftsführer von Electronic Arts (EA) Deutschland, davon aus, dass sich der Markt in den nächsten fünf Jahren verdoppeln wird. Und das “trotz Konsumflaute“ [9]. Angesichts eines erwarteten Wirtschaftswachstums von zehn Prozent ist von einer “Goldgräberstimmung der Branche“ [9] die Rede. Trotz aller Euphorie könnten Coputerspiele einen noch nicht geklärten negativen Einfluss auf ihre Teilnehmer haben. Sucht- und Gewaltfragen sind die am meisten propagierten Probleme der neuen Medien. Nach einer Umfrage glauben 90 Prozent der Befragten daran, dass Computerspiele süchtig machen können. Leider ist das Wichernhaus das bislang einzige Therapiezentrum für Mediensucht in Deutschland. Das liegt daran, dass die Computerspielsucht als Krankheitsbild noch unerforscht ist. Übliche Erscheinungen seien Hyperaktivität, Nervosität sowie Essstörungen. Peter Vorderer, Professor an der Annenberg School for Communications, sagt: "Videospiele haben die Gesellschaft längst infiltriert. Ihre Bedeutung kann gar nicht hoch genug eingeschätzt werden als Leitmedium in der Unterhaltung"[9].

Kapitel 3

Grundlagen

In diesem Kapitel werden die Grundlagen, wie die verwendete Software und die Programmiersprachen, die in dieser Arbeit Anwendung gefunden haben, erläutert.

3.1 Verwendete Software

Die Software, zu der die Programmerweiterung geschrieben werden sollte, ist "World of Warcraft". Später nur noch abgekürzt als "WoW" bezeichnet. WoW ist ein MMORPG, das bedeutet, dass mehrere tausend Spieler in einer Spielumgebung (Realm) online an einem Rollenspiel teilnehmen.

3.1.1 WoW Serverseitig

Die offiziellen WoW Server sind Cluster (Zusammenschlüsse mehrerer Server zu einem logischen Server) von bis zu 64 Servereinheiten. Als Realms werden die Spielwelten bezeichnet, die von diesen Clustern bearbeitet werden. Zurzeit können an einer Realm bis zu 3500 Spieler gleichzeitig teilnehmen. Für kommende Programmerweiterungen können die Realms noch ausgebaut werden, um das Userlimit nach oben flexibel zu halten. Da sich eine Spielwelt denkbar einfach in Kontinente, Länder und Städte aufteilen lässt, wird die Last so verteilt, dass jeder Server für eine oder mehrere Spielregionen zuständig ist. Durch diese Kapselungen lässt sich die Gesamtlast besser unter den Realmservern

verteilen. Es ist schwierig, nähere Angaben zu der serverseitig verwendeten Soft- und Hardware zu machen, da Blizzard diese Informationen aus firmenpolitischen Gründen zurückhält.

3.1.2 WoW Clientseitig

Die Clientsoftware, mit der am Spiel teilgenommen wird, ist eine 3D-Applikation. Anders als bei üblichen Computerspielen, ist das Benutzerinterface (Bedienelemente wie Fenster, Menüs, Knöpfe und alle anderen Formen von Anzeigeelementen) von den restlichen Programmteilen (3D-Engine, Net-Engine, Physik, Programmkern: Spielverlaufsroutinen, User Input/Output etc.) getrennt. Der Programmcode des Interfaces ist absichtlich offengelegt worden, um der Spielergemeinschaft die Entwicklung eigener Interface-Modifikationen (AddOns) zu ermöglichen. Alleine während der Betaphase sind die meisten der heute populären Interfaceerweiterungen zu WoW entwickelt worden. Die Vielfalt reicht hier von kleinen AddOns, die beispielsweise lediglich die Anzeige der gewonnen Erfahrungspunkte im Spiel besser darstellen, bis hin zu Applikationen, die große Datenmengen der Spielwelt entnehmen und auswerten können. Die Internetseiten von Curse-Gaming [6] und ui.worldofwar.net [8] bieten Listen der meisten aktuellen AddOns an.

3.2 WoW Userinterface

Das WoW Interface besteht aus zwei Teilen. Zum einen werden XML-Daten [2] verwendet, um Interfaceobjekte wie Fenster, Knöpfe, Rahmen, Grafiken etc. zu modellieren. Zum anderen wird die Programmiersprache LUA [11] verwendet, um das Verhalten dieser Interfaceobjekte zu definieren.

3.2.1 XML Interfaceobjekte

Die XML-Daten eines AddOns befinden sich in folgender Datei:
“<wow-directory>/Interface/AddOns/<AddOn-name>.xml“ . Zur Veranschaulichung

zeigt das folgende Beispiel, wie die XML Datei eines AddOns mit dem Namen “hello“ aussehen kann:

```
<Ui xmlns="http://www.blizzard.com/wow/ui/" xmlns:xsi="http://
   www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http:
   //www.blizzard.com/wow/ui/ C:\Projects\WoW\Bin\Interface\
   FrameXML\UI.xsd" >
  <Script file="hello.lua" />
  <Frame name="helloFrame" frameStrata="BACKGROUND"
    enableMouse="true" hidden="false" toplevel="true"
    parent="UIParent" >
    <Size>
      <AbsDimension x="200" y="20" />
    </Size>
    <Anchors>
      <Anchor point="CENTER" relativeTo="UIParent" relativePoint
         ="CENTER" />
    </Anchors>
    <Layers>
      <Layer level="ARTWORK">
        <FontString name="helloText" inherits="
           GameFontNormalSmall" text="Hallo, dies ist ein
           Beispiel AddOn!"/>
      </Layer>
    </Layers>
    <Scripts>
      <OnMouseUp>
        HELLO_OnMouseUp();
      </OnMouseUp>
    </Scripts>
  </Frame>
</Ui>
```

Hierdurch wird ein “Frame“ Element mit dem Namen “helloFrame“ erzeugt. Der Frame ist 200x20 Pixel groß und befindet sich in der Bildschirmmitte. Dieser Frame beinhaltet ein Layer mit einer Textbox namens “helloText“. Als Text wird “Hallo, dies ist ein Beispiel AddOn!“ angezeigt. Die Verbindung mit LUA entsteht durch die “<Scripts>“ Sektion. In diesem konkreten Fall wird bei dem Userevent “<OnMouseUp>“ die LUA-Funktion “ HELLO_OnMouseUp()“ ausgeführt.

3.2.2 LUA

LUA wurde nicht von Blizzard für World of Warcraft entwickelt, es ist vielmehr eine frei verfügbare Programmerweiterungs-Scriptsprache, die bereits in vielen Softwareprodukten Anwendung gefunden hat (offizielle Liste: [1]). Die 1993 in Brasilien entwickelte Programmiersprache hat Ähnlichkeiten mit den Sprachen Python [7] und Icon [4]. Da

LUA eine Bytecode-Interpretersprache ist, wird das gesamte Interface von der im Useragent integrierten LUA-VM bearbeitet. Hierfür wird zur Ladezeit der vorhandene LUA Quelltext in Bytecode umgewandelt. Die LUA Datei des Beispiel AddOns “hello“ könnte folgendermaßen aussehen:

```
function HELLO_OnMouseUp()
  -- dies ist ein Kommentar

  -- jetzt definieren wir ein lokales Array:
  local array = {"eins","zwei","drei"};
  -- so sieht eine Schleife ueber das Array aus,
  -- welche die Werte ueber den WoW Chat Frame ausgibt
  for i, v in array do
    DEFAULT_CHAT_FRAME:AddMessage(i.." = "..v);
  end

  -- Text von helloText aendern
  helloText:SetText("Sie haben das OnMouseUp Event ausgeloest");
end
```

Da in der Ausgabe eine Konkatenation des Indexes i, des Strings “ = “ und des Wertes v vorgenommen wurde, würde ein Mausklick auf den helloFrame folgende Ausgabe erscheinen lassen:

1 = eins

2 = zwei

3 = drei

Unmittelbar danach würde der Text von helloText auf “Sie haben das OnMouseUp Event ausgeloest“ geändert werden.

3.2.3 Events

Damit das Interface nicht bezugslos zu den restlichen von Blizzard festgelegten Programmteilen ist, kann es auf sogenannte Events (Ereignisse) reagieren. Diese Events werden entweder durch den Benutzer (z.B. durch den Mauscursor) oder durch den Spielverlauf (z.B. mit dem Tod des Spielcharakters) ausgelöst. Wie in Abbildung 3.1 dargestellt, werden diese beiden Typen unterschiedlich behandelt. Benutzer-Events werden direkt an das jeweilige XML-Element gerichtet (Mouseover, Drag&Drop, Resize, OnClick etc.). Spielverlaufsereignisse müssen für jedes AddOn registriert werden, damit die LUA-VM weiß, welche AddOns sie benachrichtigen muss, wenn gewisse Ereignisse ausgelöst worden sind. Es gibt ca. 360 verschiedene Spielverlaufsereignisse (Quelle:

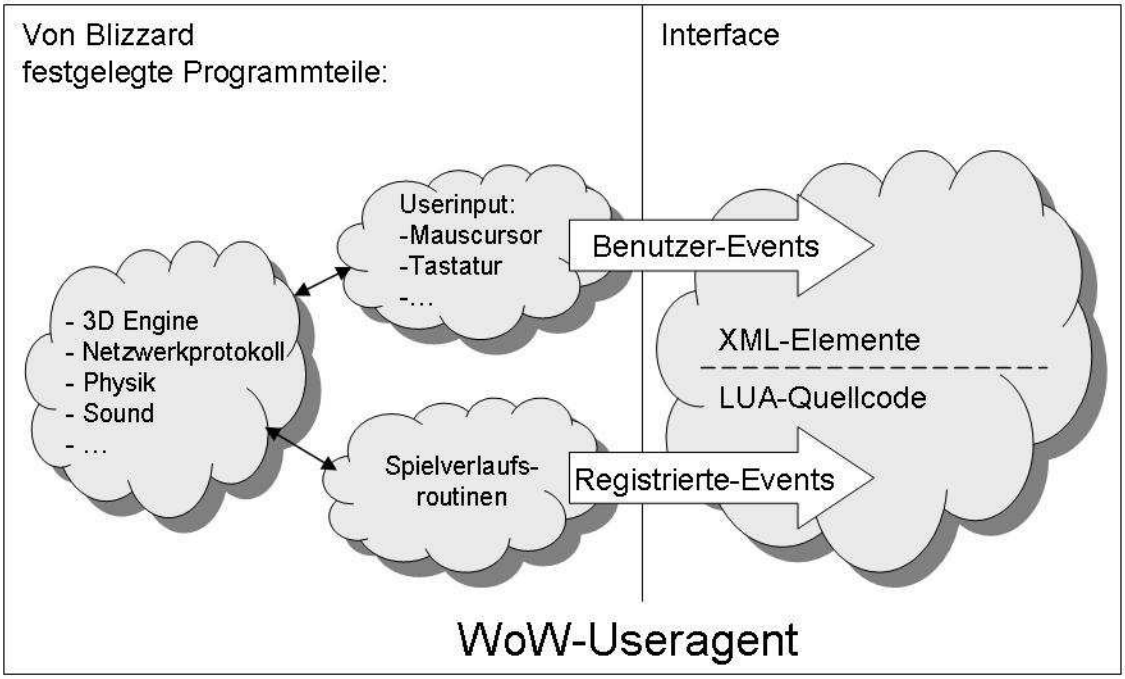

Abbildung 3.1: Diagramm - Der WoW Useragent

WoW-Wikipedia [12]). Durch die Registrierung der benötigen Events wird somit Rechenzeit gespart, da alternativ zu jedem Event alle AddOns ausgeführt werden müssten. Die Registrierung der Spielverlaufsereignisse wird üblicherweise in dem “<OnLoad>“ Handler des Frames ausgeführt, der auf diese Ereignisse mit Hilfe des “<OnEvent>“ Handlers reagieren soll. Bezogen auf das Beispiel-AddOn aus Abschnitt 3.2.1 müssen für die Registrierung eines Spielverlaufsereignisses zwei Änderungen vorgenommen werden. Als erstes muss die “<Scripts>“ Sektion des helloFrames in der XML Datei geändert werden:

```
...
    <Scripts>
      <OnMouseUp>
        HELLO_OnMouseUp();
      </OnMouseUp>
      <OnLoad>
        HELLO_OnLoad();
      </OnLoad>
      <OnEvent>
        HELLO_OnEvent();
      </OnEvent>
    </Scripts>
...
```

Danach müssen die Funktionen HELLO_OnLoad() und HELLO_OnEvent() in der LUA Datei implementiert werden:

```
function HELLO_OnLoad()
  -- Das "Der Spieler ist gestorben" Ereignis
  -- auf den helloFrame registrieren.
  this:RegisterEvent("PLAYER_DEAD");
end

function HELLO_OnEvent()
  -- Etwas Sinnvolles mit diesem Ereignis anfangen
  -- z.B. den Text von helloText aendern
  helloText:SetText("Sie sind gestorben...");
end
```

Nun würde die Anzeige des “helloFrames“ bei dem Tod des Spielers auf “Sie sind gestorben...“ geändert werden.

3.2.4 Interface API

Die Events alleine stellen keine genügende Kommunikationsgrundlage zwischen dem Programmkern und dem Interface dar. Das liegt vor allem daran, dass man mit Hilfe der Events nicht in der Lage ist, Informationen vom dem Interface an den Programmkern zu übermitteln, um diesen zu steuern. Hierzu dient die WoW Interface API. Sie ist eine Sammlung von LUA Funktionen, die zum Steuern und Abfragen des Spielverlaufs zur Verfügung stehen. Der Umfang der API variiert mit jedem neuen Patch von Blizzard. Das ist der Hauptgrund, aus dem viele AddOns nach einem Patch Fehlfunktionen aufweisen. Blizzard stellt leider ungenügend Informationen zu der API bereit. Es wurde lediglich der Quellcode des Orginalinterface zum Download bereitgestellt. Hierdurch lässt sich zwar die allgemeine Funktionsweise der meisten Funktionen erschließen, jedoch bleiben detaillierte Informationen über ungenutzte Funktionen verborgen. Derzeit umfasst die API ca. 1400 Funktionen. Um diese Funktionen zu ordnen und zu dokumentieren, existiert eine freie Wikipedia [12]. Hier kann jeder WoW Interface Programmierer Informationen nachschlagen sowie komplettieren.

3.2.5 Programmumgebung

Die Struktur eines AddOns ist durch den WoW Client folgendermaßen vorgeschrieben:

- Verzeichnisstruktur: Alle WoW Interfacemodifikationen müssen jeweils in einem eigenem Verzeichnis in “<wow-directory>/Interface/AddOns/<addon-name>“ liegen, damit der Client sie erkennt.

- TOC Datei: Jedes Interface-AddOn muss in seinem Verzeichnis eine "<addon-name>.toc“ Datei haben, welche folgende Hintergrundinformationen über das AddOn beinhaltet: Interfaceversionsnummer, Titel, Autor, Abhägnigkeiten, Variablen die bei Beendigung abgespeichert werden sollen und zu ladenden XML Dateien. Diese Daten dienen der korrekten Einbindung des AddOns während Ladezeit. Damit ein AddOn sinnvoll arbeiten kann, sollte mindestens eine XML Datei angegeben sein. Eine geeignete TOC Datei für das “hello“ AddOn sieht beispielsweise folgendermaßen aus:

```
## Interface: 1600
## Title: hello
## Notes: Ein Beispiel AddOn.
## Author: Michael Willigens
## RequiredDeps:
## OptionalDeps:
## DefaultState: enabled
## SavedVariables:
hello.xml
```

Kapitel 4

Implementierung

4.1 Anforderungen

Dieser Abschnitt widmet sich der Ausarbeitung der Anforderungen, die an ein Programm gestellt werden, welches den Spielverlauf aus der Sicht eines WoW Clients protokolliert.

- Erfassungsspektrum: Es soll möglich sein ein breites Spektrum an Informationen erfassen zu können. Die möglichen Informationen dürfen nicht vorab gewertet oder herausgefiltert werden. Die Interpretation der Daten mit Rückschluss auf Verhaltensmuster ist nicht Teil dieser Arbeit. Ereignisse bzw. Informationen die sich für eine Aufzeichnung anbieten sind:

  – Der Aufenthaltsort. Es ist sinnvoll ihn mit jedem Ereignis abzuspeichern, da sich hierdurch die Wegewahl der Spieler nachvollziehen lässt. Der zusätzlich benötigte Speicher beläuft sich lediglich auf 8 Byte pro Eintrag, da jeweils nur ein vierstelliges Koordinatenpaar abgespeichert werden muss.

  – Quests: Wann und bei wem hat ein Spieler ein Quest angenommen? Wie ist der aktuelle Fortschritt? Hat er es vielleicht nach zu langer Bearbeitung abgebrochen?

  – Charakterdaten sowie deren zeitliche Veränderung.

  - Getötete Gegner verbunden mit ihren Basisattributen (Typ: Spieler/NPC, Name, Level, Geschlecht, Rasse, Klasse, Gildenname). Hieraus lassen sich später mögliche Feindschaften erkennen.

  - Gruppenbildungen, ebenfalls mit Basisattributen der jeweiligen Gruppenpartner.

  - Der Tod des Spielers. Auch hier sind die Basisattribute der beteiligten Gegner interessant.

  - Duelle: Gegen wen und wann der Spieler ein Duell gewonnen oder verloren hat.

  - Schlachtfelder: Sie werden auch Battlegrounds genannt und bezeichnen eine abgeschlossene Spielumgebung, in der hauptsächlich PvP Kämpfe stattfinden. Die Teilnahme an einem Schlachtfeld muss aufgezeichnet werden, damit die PvP Siege oder Niederlagen anders interpretiert werden können. Das ist wichtig, um das Aggressionsverhalten korrekt analysieren zu können. In Schlachtfeldern ist jeder Spieler der anderen Fraktion ein Feind, persönliche Feindschaften lassen sich hieraus nicht ableiten, in den meisten anderen Situationen jedoch schon.

- Einbindung, Kompatibilität und Einfachheit: Das Programm sollte ohne größere Eingriffe installier- und ausführbar sein. Jede Testperson muss in der Lage sein ohne Vorkenntnisse dieses Programm auf ihrem System lauffähig zu machen. Zudem darf ein solches Programm den Spielverlauf nicht beeinflussen oder stören. Es ist denkbar, dass ein zu großer Eingriff in WoW dazu führen könnte, dass der Benutzer wegen des Einsatzes sogenannter "Third Party Tools"von Blizzard gebannt wird. Dies muss vermieden werden. Das AddOn soll als “stand alone“ Lösung umgesetzt werden. Das bedeutet, dass das AddOn keinerlei Abhängigkeiten haben darf. Trotzdem dürfen keine Inkompatibilitäten zu anderen bekannten AddOns entstehen.

- Laufzeitverhalten: World-Of-Warcraft erhebt relativ hohe Ansprüche an das Benutzerendsystem. Beispielsweise ist für einen reibungslosen Spielablauf in jeder Situation ein Gigabyte Arbeitsspeicher erforderlich. Die Anwendung darf daher

weder übermäßig viel Rechenzeit in Anspruch nehmen oder den Arbeitspeicher zu stark belasten.

- Zeitindex: Für Protokollierungsprogramme ist die Zuordnung der Ereignisse durch einen eindeutigen zeitlichen Index erforderlich. Das bedeutet auch, dass sich kein Ereignis mit einem anderen in der Reihenfolge vertauschen darf.

- Redundanzvermeidung: Ereignisse dürfen nicht mehrfach aufgezeichnet werden. Bei der LUA-VM von WoW werden gleiche Events oft mehrfach oder auch grundlos ausgelöst.

- Lokalisierung: WoW lässt sich in Englisch, Deutsch, Französisch und Koreanisch spielen. Deshalb ist es notwendig, dass sich die Applikation dynamisch an die jeweils benutzte Sprachversion von WoW anpasst. Auch die Anpassung an noch kommende Sprachversionen sollte ohne größeren Aufwand möglich sein. Damit soll nicht das Erscheinungsbild des AddOns in die jeweilige Sprache übersetzt werden, sondern die korrekte Funktionalität mit der Sprachversion sichergestellt werden. Es ist oft der Fall, dass beispielsweise auf Statusachrichten zurückgegriffen werden muss, um gewisse Informationen zu erhalten. Hier spielt die Lokalisierung die entscheidende Rolle, da diese Nachrichten für den Benutzer in der jeweiligen Sprache vorliegen.

4.2 Indexstruktur

Zunächst einmal sind für Protokollierungsprogramme Routinen erforderlich, die es ermöglichen Strings in einem zeitlich geordneten Index abzuspeichern. Weil LUA als Programmerweiterungsscriptsprache in der Wahl der Datentypen sehr eingeschränkt ist, bieten sich eigentlich nur die sogenannten “LUA-Tables“ an. Jede andere Umsetzung würde eine interne Benutzung dieser Datenstruktur bedingen, wodurch kein signifikanter Laufzeitgewinn erzielt werden könnte. Zudem erfüllen LUA-Tables alle benötigten Eigenschaften:

- Es lassen sich alle Datentypen, also auch Strings, als Variablen in einer Liste speichern.

- Es lassen sich auch LUA-Tables einbetten, wodurch eine Baumstruktur erzielt werden kann.

- LUA-Tables sind in ihrer Größe flexibel. Es lassen sich beliebig Einträge hinzufügen, entfernen oder verschieben.

- Die Indizierung erfolgt entweder numerisch oder lexikalisch.

- Eine LUA-Table lässt sich problemlos in den exportierbaren LUA Scriptspeicher schreiben.

Die genaue Implementierung der LUA-Tables ist nicht einsichtig. Es ist jedoch davon auszugehen, dass die Komplexität quadratisch mit dem Wachstum einer Tabelle ansteigt. Es ist daher sinnvoll eine geeignete Baumstruktur zu verwenden, damit nicht alle Einträge in der selben Tabelle gespeichert werden. Zudem kann durch eine geschickt gewählte Unterteilung auch redundante Information des zeitlichen Indexes gespart werden. Folgende Ebenenunterteilung wurde in dieser Arbeit implementiert: Session - Stunde - Minute - Sekunde
Session bezeichnet die aktuelle Spielsitzung, diese wird jedes Mal zur Ladezeit der LUA-VM um eins erhöht. Im Regelfall geschieht das wenn sich der Teilnehmer mit der Realm verbindet. Stunden, Minuten und Sekunden sind der mathematisch Abgerundete Wert der aktuellen Server-Uptime, jedoch relativ zum Beginn der Sitzung. Die Serverzeit wird also nur als Taktgeber verwendet. Das bedeutet, das zum Beginn jeder Sitzung die Stunde Null ist. Der Bezug zu der realen Zeit wird als Randinformation mit jeder Sitzung gespeichert. Auf der Blattebene (den Sekunden) sind die Logeinträge als Strings gespeichert. Die Funktion, welche die korrekte Verarbeitung eines abzuspeichernden Strings bearbeitet, heißt “MD_logString(string)“. Folgendes Beispiel veranschaulicht den Aufbau des Indexes:

```
[1] = { -- Session 1
  [0] = { -- Stunde 0
    [0] = { -- Minute 0
      [9] = "Logeintrag1", -- Sekunde 9
      [31] = "Logeintrag2|Logeintrag3", -- Sekunde 31, doppelter
          Eintrag
      [52] = "Logeintrag4",
    } ,
    [1] = { -- Minute 1
      [14] = "Logeintrag5",
      [3] = "Logeintrag6",
    } ,
  } ,
```

```
  ["RT"] = "21:46:43 16.06.05", -- Verweis auf die reale Zeit
  ["ST"] = 100387, -- Verweis auf die Serverzeit
} ,
```

Erläuterung:
In Sekunde 31 wurde ein doppelter Eintrag vorgenommen, weil zwei Ereignisse in der selben Sekunde geschehen sind, auf die genaue Reihenfolge ist bei der Implementierung Rücksicht genommen worden. Solche doppelten Einträge werden durch "|" (Pipe) getrennt. "RT" steht für Real-Time (Realzeit), diese wird in dem Format "HH:MM:SS dd:mm:yy" angegeben. "ST" steht für Server-Time und gibt die Server-Uptime in Sekunden an. Beide Einträge beziehen sich auf den Beginn der Sitzung. Die Server-Uptime wird wartungsbedingt jede Woche mittwochs von Blizzard durch einen Serverneustart auf Null gesetzt. Sie eignet sich am besten als Zeitgeber, da fast alle Spielverlaufsereignisse serverseitig überwacht werden und sie nicht durch eine falsch laufende Clientuhrzeit verfälscht werden kann.

4.3 Event Handling

Wird ein bestimmtes Event in WoW ausgelöst, so werden die "<OnEvent>" Handler aller XML-Elemente, die dieses Event registriert haben, aufgerufen. Während der Laufzeit des Handlers sind das Event und alle damit zusammenhängenden Informationen dann nur als globale Variablen verfügbar. Zudem ist die Sinnhaltigkeit und die zeitliche Singularität der Ereignisse in WoW durch Programmbedingte Schwierigkeiten nicht immer gewährleistet. Singularität soll bedeuten, dass ein eigentliches Spielereignis nur einmal und nicht mehrfach zu einer Eventauslösung führt. Hieraus entstehen mehrere Schwierigkeiten:

- Andere Plugins, die ebenfalls auf diese Informationen zugreifen, können diese überschreiben.

- Das Event muss erst kategorisiert und singularisiert werden, um korrekt bearbeitet werden zu können.

- Events können nur als Signal gewertet werden, dass etwas passiert ist. Die gewünschten Informationen müssen durch Benutzung der WoW Interface API (siehe

Abschnitt 3.2.4) herausgearbeitet werden.

- Der Quelltext wird unübersichtlich.

Folgendes Ablaufdiagramm veranschaulicht die Handhabung der Events, wie sie in dieser Arbeit umgesetzt wurde:

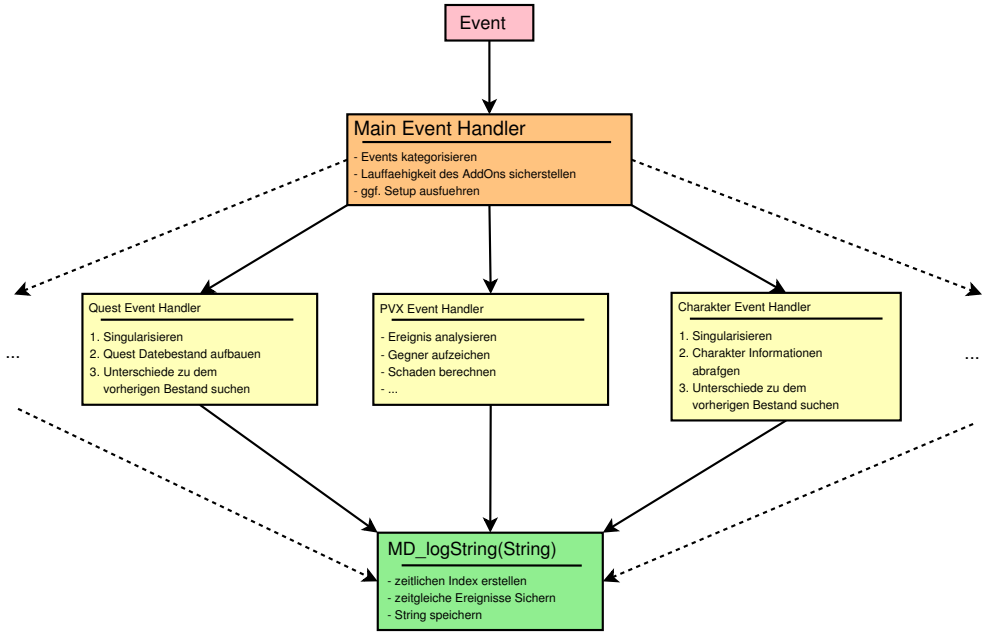

Abbildung 4.1: Ablaufdiagramm - Event Handling

Wird ein Event ausgelöst, so wird es zuerst durch den Main-Event-Handler klassifiziert. Der Main-Event-Handler ist vergleichbar mit der HELLO_OnEvent() Funktion des Beispiels aus Abschnitt 3.2.3. Sofern das AddOn lauffähig ist, ruft der Main-Event-Handler die jeweiligen Routinen auf, welche die eigentlichen Informationen aus dem Programmkern extrahieren. Diese Routinen speichern am Ende ihrer Ausführung die gewonnen Informationen durch den Aufruf der Funktion MD_logString(String). Diese erweitert den String mit dem aktuellen Aufenthaltsort des Spielers, da diese Information zu jedem Eintrag vorliegen soll. MD_logString(String) gliedert den String danach als Eintrag in den Index ein, wie er in Abschnitt 4.2 beschriebenen wurde.

4.4 Zeitlich gesteuerte Routinen

Für die Singularisierung eines Events werden zeitlich steuerbare Routinen benötigt. Jeder Aufruf der LUA-VM muss aber innerhalb eines 3D-Frames terminieren. Ein 3D-Frame bezeichnet den Zeitraum, den die 3D-Engine für die Berechnung eines Bildes benötigt. Es gibt somit keine “wait“ oder “sleep“ Funktionen, wie man dies von anderen Programmiersprachen kennt. Ein “sleep"’ würde das komplette Spiel für diesen Zeitraum stoppen lassen. Das liegt daran, dass die 3D-Engine keine neuen Bilder berechnen könnte, weil die LUA-VM noch nicht terminiert ist. Es lassen sich auch keine Threads implementieren, die über mehrere 3D-Frames hinweg LUA Code ausführen können. Für diesen Zweck wurde in dieser Arbeit ein Taskplanersystem implementiert, welches eine Tabelle mit Funktionsaufrufen sowie deren gewünschten Ausführungszeitpunkten führt. Diese Tabelle wird mit jedem dritten 3D-Frame abgearbeitet, wobei alle überfälligen Einträge ausgeführt und danach gelöscht werden. Gesteuert wird dies durch ein inhaltsloses Frame Element, welches bei dem Event “OnUpdate“, also jedes Mal wenn die 3D-Engine ein neues Bild berechnet hat, den Taskplaner ausführt. Die zeitliche Singularisierung eines Events läuft dann folgendermaßen ab:

- Das erste Event dieses Typs abfangen.
- Einen zeitlich gesteuerten Funktionsaufruf anlegen, der beispielsweise erst nach einer Sekunde (ca. 30 Frames später) dieses Event auswertet.
- Alle bis dahin eingegangenen Events dieses Typs verwerfen.

Nötig ist dieses Vorgehen vor allem bei Charakter- und Questevents, weil sie in der Regel öfter als zehn-mal Ausgelöst werden. Ohne eine Singularisierung hakt der Spielablauf bei diesen Events, da die Event-Handler entsprechend oft ausgeführt werden müssen.

4.5 Datentransfer

Natürlich müssen die gesammelten Daten irgendwann dem LUA-Scriptspeicher entnommen werden, um eine Überlastung zu vermeiden. Zum Zweck des Datentransfers ist es

möglich, den Logindex aus Abschnitt 4.2 bei dem Beenden des Spiels auf der Festplatte speichern zu lassen. Diese Daten werden dann allerdings zu der nächsten Spielsitzung wieder in den Scriptspeicher geladen. Im Rahmen der Bachelorarbeit wurde ein Werkzeug in Form einer Windows 32 Executeable in C++ entwickelt, welches diese Daten vor Beginn jeder Sitzung entnimmt. Es liest die Daten aus der LUA Scriptspeicherdatei. Diese Datei heißt “SavedVariables.lua“ und liegt pro Account in dem Verzeichnis “<wow-directory>/wtf/<account-name>/“ . Abgespeichert werden die entnommenen Logdaten in “<wow-directory>/MyDiary/<charakter-name>@<server-name>.mdd“ . Informationen über die Quests werden in der Datei mit der Endung “*.mdq“ sowie die Charakterdaten in der Datei mit der Endung “*.mdc“ gespeichert. Diese Trennung wird aus folgenden Gründen vorgenommen:

- Die Charakterdaten umfassen ein Abbild aller aktuellen Charakterwerte: Attribute, Talente, Fertigkeiten etc. Änderungen dieser Werte werden explizit in den Logdaten festgehalten. Es wäre jedoch redundant, bei jeder Änderung das komplette Charakterabbild in die Logdaten zu schreiben. Es genügt daher jedesmal nur noch ein aktuelles Abbild der Charakterdaten zur Verfügung zu haben.

- Die Questdaten umfassen Informationen über: Zeitpunkt der Annahme, NPC der das Quest anbietet, Aufenthaltsort, Questtexte, eine Quest bezogene Log die speichert was wann wo erledigt worden ist, mögliche Belohnungen usw. Die Bearbeitung eines Quest kann aber über mehrere Spielsitzungen hinweg stattfinden. Deshalb ist es sinnvoll, Quests erst nach deren Beendigung zu exportieren, damit alle Daten komplett vorliegen. Jedes Quest erhält eine eindeutige Nummer, so dass Änderungen im Fortschritt auch in den normalen Logdaten vermerkt werden können.

Sobald die Charaktere aller Accounts bearbeitet sind, wird der WoW Client automatisch gestartet. Auch das Thottbot Projekt [13] (siehe Abschnitt 2.2) benutzt diese Art des Datentransfers.

4.6 Anpassung

Das AddOn ist zwar als “stand alone“ Lösung konzipiert, es ist aber für den Benutzer von Vorteil, wenn es sich gut an jede andere denkbare Interfacemodifikation anpasst. Hierzu wurde ein grafisches Konfigurationsmenü entwickelt, welches es dem Benutzer ermöglicht, gewisse Funktionen und Anzeigen an- bzw. auszustellen. Dieses wurde dann in die gängigen Konfigurations-AddOns wie “Cosmos“ und “myAddOns“ integriert. Diese Unterstützung ist aber optional, so dass die Unabhängigkeit des AddOns nicht verletzt ist.

Kapitel 5

Testphase

In diesem Kapitel wird auf das Verhalten des AddOns eingegangen, welches durch Praxistests untersucht wurde.

5.1 Laufzeitverhalten

Auf allen Testsystemen konnte keine Einschränkung des Spielflusses durch Benutzung des AddOns festegestellt werden. Diese Tatsache bezieht sich aber nur auf die Rechenlastigkeit des AddOns. Der zusätzliche Scriptspeicher, der für die Ausführung des AddOns benötigt wird, variiert mit der Dauer der Spielsitzungen sowie damit, ob der Benutzer das beigefügte Exportprogramm benutzt. Beispielsweise wurden während acht Tagen Spielzeit eines Testcharakters zwei Megabyte Daten aufgezeichnet. Wäre das Exportprogramm nicht benutzt worden, würde der exportierbare LUA Scriptspeicher (“<wow-directory>/wtf/<account-name>/SavedVariables.lua“) so stark belastet werden, dass das Spiel störend lange Ladezeiten hätte.

Das folgende Diagramm stellt die Größe der aufgezeichneten Daten der Spielzeit gegenüber. Die Daten stammen von einem Spielcharakter der in dieser Zeit von Stufe 43 bis 49 gespielt wurde. Hierfür wurden 41 Sessions benötigt. Im Durchschnitt waren die Spielsitzungen 95 Minuten lang und enthielten 11,4 kByte an protokollierten Daten.

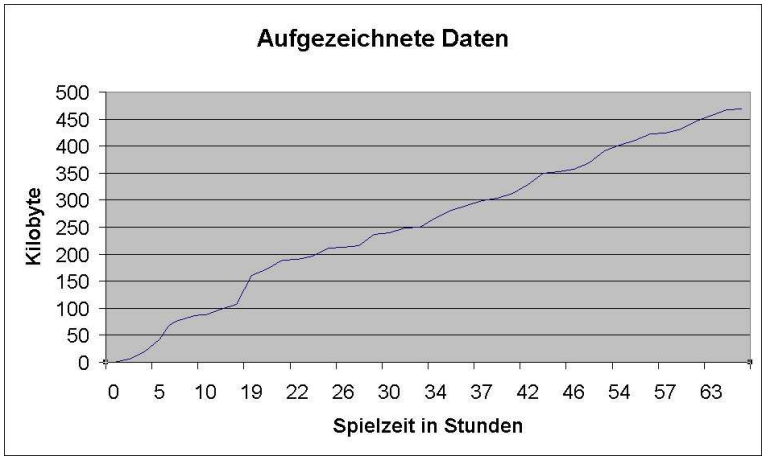

Abbildung 5.1: Diagramm - Speicherplatz/Spielstunden

Der durchschnittlich benötigte Speicher liegt also bei acht Kilobyte pro Spielstunde. Hinzu kommen Quest- und Charakterdaten die separat aufgezeichnet werden. Der tatsächliche Wert hängt dementsprechend von der Aktivität des Spielers ab, wird aber im Regelfall unter 15 Kilobyte pro Stunde bleiben. Wird der Exporter wie vorgesehen verwendet, ergibt sich durch die Verwendung des AddOns keine Einschränkung der Spielbarkeit, da exportierte Daten den Scriptspeicher nicht mehr belasten.

5.2 Auszüge

5.2.1 Log

Es folgt ein Beispiel, anhand dessen die meisten Logeinträge erklärt werden. Dieses Beispiel spiegelt eine in Wirklichkeit gespielte Situation wieder, bei der allerdings einige Einträge zu Gunsten der Übersichtlichkeit entfernt oder verändert worden sind.

```
-- Timestamp: 26.07.2005 18:34:27
[1] = {
  [0] = {
    [0] = {
      [2] = "5859,4479|ZC:1,18@Shadowglen",
      [12] = "5867,4428|QAc:1",
      [30] = "5915,4377|CSkU:Waffenfertigkeiten,Verteidigung@1
         ->2|Kill:Junger Disteleber@l:1@f:normal@d:49|CSkU:
         Waffenfertigkeiten,äStbe@1->2",
```

```
      [31] = "5915,4377|QU:1,2,1",
    },
    [4] = {
      [21] = "6220,4133|Kill:Junger äNachtsbler@l:1@f:normal@d
         :44",
      [22] = "6220,4133|QU:1,1,3|CLU:1,2|CAU:1@18->19|CAU:3@19
         ->20|CAU:4@22->23|CAU:5@22->23|CSkLU:Waffenfertigkeiten
         ,äStbe@5->10|CSkLU:Waffenfertigkeiten,Verteidigung@5
         ->10|CSkLU:Waffenfertigkeiten,Dolche@5->10|CSkLU:
        Waffenfertigkeiten,Unbewaffnet@5->10",
      [40] = "6290,4223|CSkU:Waffenfertigkeiten,Verteidigung@4
         ->5",
      [42] = "6288,4219|QU:1,1,4|CSkU:Waffenfertigkeiten,äStbe@4
         ->5",
      [55] = "6344,4189|QU:1,1,5",
      [54] = "6339,4196|Kill:Junger äNachtsbler@l:1@f:normal@d
         :46",
      [55] = "QU:1,1,7|QC:1",
    },
    [6] = {
      [2] = "5984,4259|QAc:3",
      [6] = "5945,4320|ZC:1,18@Shadowglen",
      [35] = "5867,4432|QAc:5",
      [58] = "5778,4530|QF:3",
      [24] = "5867,4432|QF:1",
      [33] = "5867,4432|QAc:4",
      [59] = "5778,4530|QAc:6",
    },
    [7] = {
      [46] = "5865,4437|DW:Zagrr@l:2@r:Orc@c:Krieger@s:0",
      [55] = "5868,4433|PJ:Zagrr@l:2@r:Orc@c:Krieger@s:0",
    },
  },
  ["RT"] = "18:26:01 26.07.05",
  ["ST"] = 16896.6,
},
```

Auf der ersten Ebene des Indexbaumes befinden sich die Nummern der Sitzungen. In diesem Fall fanden alle Einträge in der Sitzung mit der Nummer eins statt. Es handelt sich also um die erste Sitzung dieses Charakters. Der Zeitpunkt, an dem die Sitzung angelegt wurde, ist der 26. Juli 2005 um 18:26 Uhr (gekennzeichnet durch den [“RT“] Eintrag). Der Server lief zu diesem Zeitpunkt seit 16896.6 Sekunden (gekennzeichnet durch den [“ST“] Eintrag). Da sich der Index wie in Abschnitt 4.2 beschrieben liest, werden ab jetzt nur noch die Zeitpunkte genannt und erklärt, was dort passiert ist. Auch auf das Koordninatenpaar, das vor jedem Logeintrag steht, wird nicht näher eingegangen.

- Sekunde 2: “ZC“ (Zone-Change). Ein Zonenwechsel wurde festgestellt. Da das Betreten der Spielwelt auch als Zonenwechsel interpretiert wird, ist dies der erste Eintrag. “1,18@Shadowglen“ bedeutet, dass auf den ersten Kontinent auf die 18. Zone gewechselt wurde. Eine Zone kann als Landkarte aufgefasst werden. “Shadowglen“ ist die sogenannte Subzone, sie bezeichnet den Ort innerhalb der aktuellen Zone genauer.

- Sekunde 12: “QAc“ (Quest-Accepted). Es wurde ein Quest von einem NPC angnommen. In diesen Fällen gibt die folgende Nummer die ID des Quests an, welches angenommen wurde.

- Sekunde 30: Hier sind drei Einträge in der selben Sekunde aufgezeichnet worden. “CSkU“ (Character-Skill-Update). Ein Talent des Spielcharakters hat sich geändert. In diesem Fall wurde die Verteidigungsfertigkeit von eins auf zwei erhöht. Der Eintrag “Kill:Junger Disteleber@l:1@f:normal@d:49“ zeigt an, dass ein Gegner getötet wurde. Es handelt sich hierbei um ein Level eins Monster, das mit 49 Schadenspunkten bezwungen worden ist. Danach folgt ein weiteres “Character-Skill-Update“.

- Sekunde 31: “QU“ (Quest-Update). Das bedeutet, das sich etwas an dem Bearbeitungsstatus eines Quest geändert hat. Hier wurde für die zweite Teilaufgabe des ersten Quests die erste Änderung vermerkt. Die genauen Informationen können aus der Questlog entnommen werden. Dieses Update hängt mit der Tötung des Monsters aus Sekunde 30 zusammen.

- Minute 4, Sekunde 22: “CLU“ (Character-Level-Update). Das bedeutet, dass der Spieler die nächste Stufe erreicht hat, wodurch eine Reihe von “Character-Skill-Update“ und “Character-Attribute-Update“ (“CAU“) Ereignisse festgestellt worden sind.

- Minute 4, Sekunde 55: “QC“ (Quest-Complete). Der Bearbeitungsstatus des Quests mit der ID eins ist komplett, und kann somit bei einem NPC abgegeben werden.

- Minute 6, Sekunde 24: “QF“ (Quest-Finished). Das Quest mit der ID eins wurde abgegeben.

- Minute 7, Sekunde 46: “DW“ (Duel-Win). Das bedeutet, dass ein Duell gewonnen

wurde. Hier wurde der Spieler “Zagrr“, ein männlicher Level zwei Orc-Krieger, besiegt.

- Minute 7, Sekunde 55: “PJ“ (Player-Joined). In diesem Fall ist der Spieler, gegen den gerade das Duell gewonnen wurde, der Party beigetreten ist.

5.2.2 Quests

Quests werden, auf Grund ihrer langen Bearbeitungszeit, in einer separaten Logdatei gespeichert. Im Folgenden wird der Auszug eines aufgezeichneten Quests gezeigt, welches komplett bearbeitet wurde. Die Anzahl der Aufgaben, sowie die Beschreibungen dieser, wurden nachträglich verkürzt.

```
  [21] = {
    ["eSession"] = 5,
    ["eZone"] = 12,
    ["aNpc"] = "Hexendoktor Uzer'i",
    ["flag"] = "Elite",
    ["isCompleteable"] = true,
    ["aZone"] = 12,
    ["eNpc"] = "Hexendoktor Uzer'i",
    ["objectives"] = {
      [1] = {
        ["logs"] = {
          [1] = "14:43:22 09.07.05-1,12-4158,2566|Bergriesen-
             Muisek: 1/4",
          [2] = "14:45:05 09.07.05-1,12-4127,2609|Bergriesen-
             Muisek: 2/4",
          [3] = "14:46:48 09.07.05-1,12-4130,2545|Bergriesen-
             Muisek: 3/4",
          [4] = "14:48:27 09.07.05-1,12-4011,2556|Bergriesen-
             Muisek: 4/4",
        },
        ["type"] = "item",
        ["title"] = "Bergriesen-Muisek: 4/4",
        ["done"] = 1,
      },
    },
    ["eContinent"] = 1,
    ["isClean"] = true,
    ["numchoiceitems"] = 0,
    ["status"] = "finished",
    ["level"] = 50,
    ["aTime"] = "14:34:50 09.07.05",
    ["numrewarditems"] = 0,
    ["aSession"] = 4,
    ["ePos"] = {
      ["y"] = 4345,
```

```
      ["x"] = 7440,
    },
    ["eSubzone"] = "Camp Mojache",
    ["group"] = "Feralas",
    ["money"] = 0,
    ["aPos"] = {
      ["y"] = 4346,
      ["x"] = 7440,
    },
    ["numrequireditems"] = 0,
    ["eTime"] = "15:03:11 09.07.05",
    ["numobjectives"] = 1,
    ["objectivestext"] = "Schaltet 4 Landgaenger oder
       Klippenriesen aus. Benutzt das Muisek-Gefaess, um ...",
    ["title"] = "Bergriesen-Muisek",
    ["aContinent"] = 1,
    ["doExport"] = true,
    ["description"] = "Bei der letzten Aufgabe, die ich Euch
       stelle, sollt Ihr nach Nordwesten reisen und den letzten
       Feind finden...",
    ["logid"] = 18,
    ["aSubzone"] = "Camp Mojache",
  },
```

Erläuterung:
Alle Informationen wurden in einer LUA Table gespeichert. Der Name dieser LUA Table ist "21". Das ist die eindeutige Nummer für dieses Quest. Kein weiteres Quest wird diese Nummer bekommen. Die einzelnen Felder haben folgende Bedeutung, wobei hier nur auf die wichtigsten Einträge eingegangen wird:

- eSession - Die Nummer der Spielsitzung in der das Quest beendet wurde.
- eZone - Die Nummer der Zone (hier 12, das Land Feralas) in der das Quest abgegeben wurde.
- aNpc - Der Name des NPCs, bei dem das Quest angenommen wurde.
- flag - Dieser String kann die Werte “Elite“, “Dungeon“ oder “PVP“ haben und gibt damit an, von welchem Typ dieses Quest ist.
- aZone - Die Nummer der Zone in der das Quest angenommen wurde.
- eNpc - Der Name des NPCs, bei dem das Quest abgeschlossen wurde.
- objectives - Eine eingebettete LUA Table, welche die genauen Fortschritte der

jeweiligen Aufgaben eines Quests enthält. Hier wird gespeichert wann und wo Fortschritte in einem Quest erzielt wurden. Hierzu verfügt diese LUA Table unter [“logs“] einen numerischen Index, in welchem die Ereignisse zu den jeweiligen Aufgaben gespeichert werden.

- eContinent - Die WoW Spielwelt besteht aus zwei Kontinenten. eContinent gibt die Nummer des Kontinents an, auf dem das Quest abgeschlossen wurde.

- isClean - Ein boolean Wert, der angibt, ob das Quest regulär aufgezeichnet worden ist. Beispielsweise könnte ein Quest angefangen haben, bevor das AddOn zum Einsatz kam, in diesem Fall wären die Informationen zu diesem Quest nicht komplett. Dann wäre “isClean“ false gesetzt.

- status - Ein String, der die Werte “finished“, “open“ oder “aborted“ annehmen kann und damit den aktuellen Status des Quests angibt.

- aTime - Ein Zeitstring, der die Zeit angibt zu der das Quest angenommen wurde.

- aSession - Die Nummer der Spielsitzung in der das Quest angenommen wurde.

- aPos - Das Koordinatenpaar der Position an dem das Quest angenommen wurde.

- ePos - Das Koordinatenpaar der Position an dem das Quest abgeschlossen wurde.

- title - Der Titel des Quests.

- aContinent - Die Nummer des Kontinents, auf dem das Quest angenommen wurde.

- doExport - Dieser boolean Wert teilt dem Exportprogramm mit, ob es das Quest exportieren soll. Quests werden in der Basiskonfiguration noch über 10 Spielsitzungen in dem Scriptspeicher gehalten, um die Informationen noch erreichbar zu haben.

- description - Der String, den der Questgeber NPC als Beschreibung zu dem Quest angegeben hat.

Kapitel 6

Ausblick

Wie die Ergebnisse aus Kapitel vier gezeigt haben, stellt diese Arbeit eine zuverlässige Grundlage für das Sammeln von WoW Spielverlaufsdaten dar. Die Funktionalität des AddOns ist problemlos erweiterbar, so das es an die Bedürfnisse kommender Untersuchungen angepasst werden kann. Der nächste logische Schritt ist es nun, eine Datenbank zu erstellen, die für das Erfassen dieser Daten angepasst ist. Hierzu wird ebenfalls ein Parser benötigt, der diese Daten beispielsweise in SQL Code umwandelt, bevor der Abgleich mit der Datenbank stattfinden kann. Erst nach diesem Arbeitsschritt können diese Daten von vielen Testpersonen gesammelt und unter psychologischen sowie soziologischen Gesichtspunkten ausgewertet werden. Durch Data Minig könnten hier bislang unbekannte Zusammenhänge aufgedeckt werden. “Data Mining ist die Gewinnung impliziter, bislang unbekannter und potenziell nützlicher Informationen aus Daten“ [10]. Es handelt sich dabei um eine Arbeit bei der Psychologen und Informatiker gleichermaßen zusammen arbeiten müssten. Um das Spektrum an Testpersonen zu erweitern, lassen sich die gesammelten Daten auch für den Teilnehmer nutzbar machen.

Beispielsweise könnten diese Daten zu der Erstellung persönlicher Tagebücher verwendet werden. Es ist zu erwarten, dass sich einige Gilden für die Tagebücher ihrer Mitglieder interessieren. Es wäre möglich, da sich die meisten Gilden über Foren und Internetseiten organisieren, diese Tagebücher PHP-, Java- oder schlicht HTML-basiert zur Verfügung zu stellen.
Es gibt auch die Möglichkeit einige der gesammelten Daten spielintern zu verwenden. Inzwischen benutzen einige AddOns die internen Chat-Kanäle, um Daten mit anderen Clients auszutauschen. Die gesammelten NPC und Quest Daten könnten über den

gleichen Weg mit anderen Teilnehmern geteilt werden. Hierdurch könnten die Spieler viel gezielter ihre Quests angehen, da sie direkt wüssten wo sie was zu erledigen haben, um ein Quest erfolgreich abzuschließen. Aber nicht nur das Teilen von Questinformationen ist sinnvoll, auch Informationen über den aktuellen Aufenthaltsort von Spielern der anderen Fraktion sind Daten, die unter den Clients ausgetauscht werden könnten.

Literaturverzeichnis

[1] Offical Lua project list. http://lua.org/uses.html, 1994-2005.

[2] Extensible Markup Language. http://www.w3.org/XML/, 1996-2003.

[3] Projekt abrufbar über http://www.mmogchart.com, 2002-2005.

[4] The Icon Programming Language. http://www.cs.arizona.edu/icon/, 2003.

[5] Projekt abrufbar über http://warcraftrealms.com, 2005.

[6] Curse-Gaming. Multigaming Community - http://curse-gaming.com, 2005.

[7] Python Programming Language. http://www.python.org, 2005.

[8] The Unoffical WoW UI Site. http://ui.worldofwar.net, 2005.

[9] Markus Deggerich. Kulturkampf im Kinderzimmer. *Der Spiegel*, 33:40–42, August 2005.

[10] Eibe Frank Ian H. Witten. *Data Mining - Praktische Werkzeuge und Techniken für das maschinelle Lernen*. Hanser Verlag 2001, 2001.

[11] Luiz Henrique de Figueiredo Roberto Ierusalimschy, Waldemar Celes. Lua, the programming language. Offical Webpage http://lua.org, 1994-2005.

[12] Rustak. World Of Warcraft - Wikipedia. Projekt abrufbar über http://wowwiki.com, 2005.

[13] Thott. Thottbot. Projekt, abrufbar über http://thottbot.com, 2004-2005.

[14] Nick Yee. The Daedalus Gateway. Projekt, abrufbar über http://nickyee.com/daedalus/gateway_intro.html, 2003-2005.

[15] Nick Yee. The Daedalus Project. Projekt, abrufbar über http://nickyee.com/daedalus/, 2003-2005.

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmitteln benutzt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, den 17. August 2005 Michael Willigens