# Technisches zu "Die letzten Tage von Aymargeddon"

Aymargeddon Development Team

9.10.2003

## 1 Generelles Design

Das Spiel besteht aus folgenden Komponenten:

1. Eine Relationale Datenbank
2. Ein Dämonprozess im Server
3. Serverseitige Scripten zur Benutzerinteraktion
4. Ein Weltengenerator
5. FROGS
6. Ein Programm, dass die Integrität der Datenbank überprüft
7. Eine Bibliothek für gemeinsame Funktionalität

Die Aufgaben dieser Komponenten stellen sich wie folgt dar:

## 2 Datenbank

In dieser Datenbank wird der Zustand aller Spielwelten gespeichert. Außerdem alle Spielerdaten, alle Spieleraktionen und alle Nachrichten an Spieler. Sie sorgt mittels ihrer Transaktionen dafür, dass auch bei konkurierendem Zugriff die Datenintegrität immer erhalten bleibt.

Wir verwenden MySQL zur Implementierung und PhpMyAdmin zur Administrierung der Datenbank.

Felder, die in vielen Tabellen vorkommen und immer wieder das selbe bedeuten:

**GAME** Das ist die Id des Spiels. Dadurch können alle Spiele in der selben Datenbank verwaltet werden. Es kann maximal max(unsigned smallint) Spiele gleichzeitig geben.

**LOCATION** Feldkoordinaten auf dem Hexraster-Torus. Ein String der Form <x>_<y>. Die maximale Größe der Welt ist max(unsigned smallint) für die Y-Koordinate und max(unsingend smallint)*2 für die X-Koordinate.

**PLAYER,OWNER,...** Spieler-Ids. Die Spieler-Ids bezeichnen den Spieler *unabhängig* von seiner Rolle. Sie gilt spielübergreifend. Die maximale Anzahl Spieler ist auf max(signed smallint) beschränkt.

## 2.1 Bewegung und Kämpfe

Einheiten, die sich bewegen, bleiben im Feld stehen, werden aber auf nicht available gesetzt. Am Ende des Befehls werden sie in das neue Feld gesetzt. Sie werden nur wieder aktiv, nachdem alle denkbaren Kämpfe ausgeführt wurden.

Kämpfe werden als Quasi-Befehl wieder in die Befehlsqueu geschrieben. Erst nach Ablauf dieses Quasi-Befehls wird ausgewertet, welche Einheiten auf welcher Seite am Kampf teilnehmen.

Einheiten, die sich zurückziehen, werden ganz normal bewegt.

## 2.2 Tabelle MAP

Das ist die zentrale Karte. Für jedes Feld in jedem Spiel gibt es genau einen Eintrag.

**HOME** Eigentümer der Heimatstadt. Das Feld ist -1, wenn es eine Heimatstadt ist, aber noch niemand spielt.

**OCCUPANT** Besitzer des Feldes

**TERRAIN** kann sein eins aus: WATER, CITY, MOUNTAIN, ISLE, PLAIN

**PLAGUE** ist das Feld verseucht? Kann eine aus einer Liste von Seuchen sein

**ATTACKER** Hier steht der leitende Erdling eines Angriffs drinnen so lange gekämpft wird. 0 sonst. Man kann hier also auch ablesen, ob das Feld umkämpft ist.

**LAST_PRODUCE** Zu dieser Zeit wurde zu letzt ein Krieger (bei Städten) bzw. ein Priester (bei Tempeln) produziert. Der Dämon entscheidet anhand dieser Daten, wann neue Einheiten produziert werden.

**FLUXLINE** Hier stehen die Richtungen, in die sich Avatare momentan kostenlos bewegen können. Die benachbarten Richtungen kosten 1 MP, alle anderen 2MP. Dieses Feld wird bei einer Änderung der IdS für die gesamte Karte neu berechnet.

**TEMPLE** Steht auf 'Y', wenn dort ein Tempel gebaut wurde, auf 'N' sonst.

## 2.3 Tabelle MOBILE

In dieser Tabelle werden alle beweglichen Objekte abgespeichert. Das sind also zunächst: Krieger, Helden, Priester, Avatare und Archen. Dabei gibt es nur einen Eintrag für gleichartige Einheiten im selben Feld im selben Spiel.

Manche Felder werden nur für manche Objekttypen benutzt. Hier wird also ein bisschen Speicherplatz geopfert um die Struktur möglichst einfach zu halten.

**ID** Eine eindeutige ID.

**TYPE** Ist einer aus WARRIOR, HERO, PRIEST, AVATAR, ARK

**OWNER** Der Spieler, der die Einheit steuert

**ADORING** Der Gott, den der Priester anbetet

**COUNT** Anzahl

**AVAILABLE** Wird auf 0 gesetzt, wenn die Einheit beschäftigt ist (sich also z.B. bewegt)

**STATUS** Eines aus HELP, BLOCK, PEACE. Avatarstatus.

## 2.4 Tabelle COMMAND

In diese Tabelle tragen die Scripten die Aktionen der Spieler ein und der Dämon führt diese dann aus. Zusätzlich kommen hier auch noch die Quasi-Befehle des Dämons selber rein. Das ist alles, wo er sich für später dran errinnern will. Zur Zeit wird dieser Mechanismus nur für Kämpfe benötigt.

**TIME** Die Zeit zu der das Kommando eingetragen wurde

**ACK** Hier wird vermerkt, dass der Dämon das Kommando zur Kenntnis genommen hat, aber noch nicht ausgeführt. Das ist nötig weil bei vielen Kommandos schon am Anfang Nachrichten generiert werden müssen, lange bevor sie ausgeführt werden. Z.B. erhalten die Eigentümer eines Feldes, in das man sich bewegt, eine Nachricht, schon wenn man sich auf den Weg macht.

## 2.5 Tabelle MESSAGE

In diese Tabelle trägt der Dämon Nachrichten an die Spieler ein und die Scripten zeigen diese dann an. Nachrichten an Alle Spieler müssen für jeden Spieler einzeln eingetragen werden. Wenn man es anders machen wollte, müsste man wiederum für jeden Spieler vermerken, welche Nachrichten er nicht mehr sehen will, was fast auf das selbe rauskommt.

**TIME** Der Zeitpunkt, an dem die Nachricht generiert wurde

**FROM** Der Absender. 0 bedeutet, dass es eine automatisch generierte Nachricht des Dämon ist.

**TO** Der Empfänger

**TYPE** Message, Error, Warning, ...

**MSG** Die eigentliche Meldung. Bzw. ein Tag, dass erst noch lokalisiert werden muss (Siehe Tabelle LOCALIZE)

**ARG1...4** Die Argumente für die Lokalisierung.

## 2.6 Tabelle GAME

Hier stehen allgemein Infos das Spiel betreffend. Pro Spiel gibt es nur einen Eintrag.

**SIZE** Die Größe des Spiels. Höhe und halbe Breite des Spielfeldes. Maximale Anzahl Erdlinge.

**FORTUNE** Der Glücksfaktor

**LAST_TEMPLE** Die LOCATION des letzten fertig gestellten Tempels.

**TEMPLE_SIZE** Größe des nächsten Tempels.

## 2.7 Tabelle PLAYER

Hier wird spielunabhängig gespeichert, was es alles über einen Spieler zu wissen gibt. Pro Spieler ein Eintrag.

## 2.8 Tabelle ALLIANCE

Hier wird beschrieben welche Freunde und Feinde man hat. Pro Spieler-Spieler-Relation in jedem Spiel höchstens ein Eintrag. Status kann sein "FRIEND", "FOE" oder "NEUTRAL". Wenn kein Eintrag vorhanden ist, wird neutraler Status angenommen.

Man beachte dass Spieler A, Spieler B als Freund ansehen kann, wärend umgekehrt Spieler B Spieler A als Feind betrachtet!

## 2.9 Tabelle GOD

Hier werden Daten für die Götter gespeichert. Pro Gott und Spiel ein Eintrag.

**DEATH_AVATAR** Die Anzahl der für diesen Gott in diesem Spiel gestorbenen Avatare

**DEATH_HERO** dsgl. für Helden

**ARRIVAL** Hier entstehen neue Avatare. Dieser Ort wird nach jedem Tempelbau diesen Gottes neu berechnet.

### 2.10 Tabelle LOCALIZE

Mit Hilfe dieser Tabelle kann die Darstellung in verschiedenen Sprachen erfolgen.

**TAG** Der Eintrag mit dem man wiederkennt, um welche Message es sich handelt

**LANG** Die Sprache des Eintrags. Zur Zeit werden nur "DE" und "EN" unterstützt.

**TEXT** Der Text der Nachricht in den einzelnen Sprachen. Dabei wird mittels "%n" das n.te Argument eingefügt. "%%" gibt ein Prozentzeichen aus.

### 2.11 Tabelle ROLE

Hier wird die Rolle eines Spielers in einem Spiel beschrieben. Pro Mitspieler in jedem Spiel ein Eintrag.

## 3 Dämon

Dieses Programm liest Spieleraktionen aus der Datenbank, berechnet die sich daraus ergebenden Ereignisse und schreibt Nachrichten an die Spieler zurück in die Datenbank.

Wir verwenden Perl 5.8 zur Implementierung des Servers.

## 4 Scripten

Sie lesen den Zustand der Welt und die Nachrichten aus der Datenbank, halten Session-Informationen vor und bereiten dies alles in HTML zur Darstellung mittels eines üblichen Web-Browsers auf. Schließlich schreiben sie die Aktionen des Benutzers in die Datenbank und verändern den Aktivitätsstatus von beweglichen Einheiten.

Wir verwenden EmbPerl auf Apache zur Implementation. Siehe: http://perl.apache.org/embperl/. EmbPerl scheint genauso einfach und schnell zu sein wie PHP und hat für uns den zusätzlichen Vorteil, dass wir gemeinsame Bibliotheken mit den anderen Komponenten des Servers benutzen können.

### 4.1 Seiten

Folgende Seitenlayouts werden benötigt. Auf allen Seiten findet man ein Hauptmenu. Auf Login und Home gibt es auch noch ein Aymaegeddon-Banner

**Login** Hier gibt es neben News einen kurzen Einleitungstext sowie eine Möglichkeit sich zu registrieren und mal in einem Fakespiel zu schnuppern.

**Home** Liste aller Spiele, pro Spiel: Liste aller Nachrichten, aller Ereignisse, Statistik

**Karte** Aktuelles Feld, Beschreibung, Befehle

**Spieler** Beschreibung des Spielers

**Rolle** Beschreibung der Rolle

Feldnamen/-koordinaten sind überall immer zur Karte mit dem Feld als aktuellem Feld verlinkt. Rollennamen sind zu der entsprechenden Rollenseite verlinkt.

## 4.2 Karte

Zentrale Komponente der Darstellung ist eine Karte des Hex-Torus. Dazu werden 3 Tabellenzellen pro Feld verwendet, nämlich so:
<BILD FEHLT>

Diese Karte ist scrollbar. Ein Feld ist immer als aktuelles Feld umrandet.

## 4.3 Farbdarstellung

Wasserfelder blau, Landfelder, Archen und Inseln in Erdfarben. Dabei gibt es 5 Farbtöne für eigene, befreundete, neutrale, feindliche sowie unbesiedelte Felder. Tempel und Avatare werden in 5 verschiedenen Götterfarben (eher grell) dargestellt, wieder je eine für eigene, befreundete, feindliche sowie neutrale Götter. Die "eigene" Farbe kann auf andere Erdlinge/Götter verändert werden.

## 4.4 Icons

Folgende Icons werden benötigt.
Zentriert:

- IdS
- Eigentum auf Wasser (Schiff)
- Städte
- Tempel
- Heimatstädte
- Inseln
- Berge

Nicht zentriert:

- Avatare (oben bis zu vier)
- Archen (unten, nur eine)
- Kampf (unten)
- Avatarkampf (oben)

# 5 Weltengenerator

Dieses Programm wird einmal zu Beginn eines neuen Spiels aufgerufen um eine neue Welt in der Datenbank zu generieren. Der Generator verteilt die verschiedenen Geländetypen: Wasser, Manapol, Insel, Berg, Stadt, Heimatstadt, Land. Er erhält die Anzahl der Erdlinge als Parameter und ermittelt alle anderen Werte daraus.

Die Game-ID kann automatisch als die erste Freie in der DB ermittelt werden. Dieses Programm sollte als erstes entwickelt werden, damit man eine sinnvolle Testumgebung für die anderen Teile des Systems hat.

Wir verwenden Perl 5.8 zur Implementation.

# 6 FROGS

FROGS steht für **F**ramework for **R**ealtime **O**nline **G**ames of **S**trategy. Dort werden alle Funktionalitäten versammelt, die nicht nur von Aymargeddon, sondern auch von anderen Browser-MMOGs verwendet werden können. Das sind im einzelnen:

- Nachrichtenverwaltung
- Befehlsverwaltung
- Spielerverwaltung
- Spielverwaltung
- Rollenverwaltung
- Verschiedene Standardkarten (hier erstmal nur Hextorus)
- Bewegliche Einheiten
- Lokalisierung
- Sessionhandling
- Bestenlisten
- Datenbank

FROGS basiert dabei auf der Annahme, dass bestimmte Felder in bestimmten Tabellen vorhanden sein müssen. Außerdem werden die konkreten Funktionalitäten über Hooks in das Framework eingehängt. So wird z.B. für jeden Befehl ein Name festgelegt mit drei Hooks:

**test** Diese Funktion tested, ob der Befehl überhaupt ausführbar ist und wird noch in der direkten Benutzerinteraktion von den Scripten ausgeführt.

**ack** Diese Funktion wird ausgeführt, wenn der Befehl zum ersten mal vom Dämon zur Kenntnis genommen wird.

**do** Diese Funktion führt schließlich den Befehl aus. Dazu sind am Anfang noch weitere tests nötig.

Ziel für Frogs ist, dass man relativ einfach neue Browserspiele bauen kann. Es wird auch ein Satz von Standardseiten in EmbPerl mitgeliefert mit denen Funktionen wie Einloggen, Spielverwaltung, Bestenlisten etc. schon vorhanden sind.

Hier noch eine Liste von FROGS-Modulen und was sie tun sollen:

**Map.pm** Dies ist eine Basisklasse für alle denkbaren Topologien. Jedes Modul einer abgeleiteten Klasse sollte auch eine Klasse Location zur Verfügung stellen. Außerdem müssen abgeleitete Klassen einige Funktionen mitbringen, damit die in Map vorhandenen Funktionen funktionieren.

**HexTorus.pm** Dies ist die von Aymargeddon verwendete Topologie. Kann aber auch von anderen Spielen verwendet werden. Abgeleitet von Map.pm. Stellt auch die Klasse Location zur Verfügung.

**Checker.pm** Hier werden die verallgemeinerbaren Funktionen des Checkers zur Verfügung gestellt.

**Scheduler.pm** Hier wird die Befehlsqueu durchgegangen und die oben definierten Funktionen werden aufgerufen.

**Localize.pm** Hier wird die Lokalisierung ausgeführt.

**DataBase.pm** Hier werden Basisdatenbankfunktionalitäten zur Verfügung gestellt

**...** weitere Module noch unklar

Auch FROGS wird in Perl 5.8 bzw. EmbPerl implementiert.

# 7 Checker

Dieses Programm überprüft, ob die Daten in der Datenbank noch konsistent sind. Dabei werden die Checks zu algorithmisch ähnlichen Gruppen zusammengefasst und durch allgemein Funktionen ausgeführt. Bisher sind folgende Funktionen identifiziert worden:

1. Jeder Eintrag in Tabelle X muß auch in Tabelle Y existieren.
2. N Einträge in der selben Tabelle müssen eine logische Beziehung erfüllen

Diese allgemeinen konfigurierbaren Check-Funktionen sollten auch Teil von FROGS werden.

Der Checker überprüft im einzelnen (Zahlen beziehen sich auf obige Funktionsliste):

- Jede Spielnummer muß in der Tabelle GAME zu finden sein (1).
- sämtliche Spieler-IDs müssen in ROLE zum selben Spiel passen (1).
- sämtliche Spieler-IDs müssen in PLAYER vorhanden sein (1).
- Location muß immer in MAP vorhanden sein.
- Location muß immer die kanonische Form haben (2).
- HOME nur gesetzt in MAP, wenn TERRAIN = CITY (desgl. für GOD_HOME und MOUNTAIN) (2).
- Keine Zwei Erdlinge im selben Feld, außer es ist Kampf.
- Alle Einheiten in COMMANDS müssen inaktiv sein.
- Nur Priester ADORING in MOBILE (2).
- AVAIABLE immer kleiner oder gleich COUNT in MOBILE (2)
- Während eines Kampfes nur aktive Erdlinge eines Spielers im selben Feld.
- Keine blockenden Avatare von zwei feindlichen Spielern im selben Feld ohne Kampf .
- Jedes Tag in MESSAGE sollte in LOCALIZE vorhanden sein. Mindestens in einer Sprache. Warnung, wenn nicht in jeder Sprache.
- Die Anzahl der Argumente in MESSAGES sollte mit den nicht doppelten %-Zeichen in LOCALIZE übereinstimmen (für jede Sprache).

Dieses Programm sollte möglichst früh entwickelt werden, da es vor allem im Entwicklungsprozess benötigt wird.

Wir verwenden Perl 5.8 zur Implementation.

# 8 Bibliothek

Hier werden alle Funktionalitäten versammelt, die von mindestens zwei der Komponenten (Scripten, Generator, Dämon, Check) verwendet werden.

Dabei bleiben in dieser Bibliothek nur Sachen, die nicht noch allgemeiner sind und somit in den FROGS-Teil gehören. Momentan ist noch unklar, ob da überhaupt was übrig bleibt.

Wir verwenden Perl 5.8 zur Implementation.

# 9 Copyright