

Hinweise zum Client/Server Protokoll - V1.0

Mattias Gärtner

April 17, 2007

Abstract

Hinweise zu den Nachrichtenklassen und deren Abfolge

1 Überblick

Die Nachrichtenklassen sind fest vorgegeben, um Kompatibilität zwischen den Gruppen zu schaffen. Dennoch enthalten sie Spielraum für eigene Definitionen. Sie sind in einem Java Package namens pp07.DungeonMessages zusammengefasst und sind auf der Webseite des Lehrstuhls zum Herunterladen hinterlegt.

Die Klassen unterstützen mehrere Spielwelten, Accounts, Ein- und Ausloggen, Erzeugen und Löschen von Charakteren, Ganz- und Teilaktualisierung des Levels und Aktionen.

1.1 Entwarnung

Die Nachrichtenklassen unterstützen mehr, als von den Gruppen verlangt wird. Einige Teile, wie mehrere Spielwelten und mehrere Accounts können implementiert werden, sind aber nicht notwendig zum Bestehen.

1.2 Eine Sitzung aus Sicht des Clients

Wenn der Client, bzw. die GUI gestartet wird, loggt er sich beim Server ein und startet einen Charakter. Der Server sendet daraufhin Leveldaten. Der Client macht eine oder mehrere Aktionen, während parallel der Server neue Leveldaten schickt. Irgendwann wird der Client sich ausloggen.

1.3 Aus Sicht des Servers

Aus Sicht des Servers verhalten sich die Clients wie ein Taubenschlag. Ein ständiges Kommen und Gehen, und jeder Client hinterlässt Dreck. Wenn ein Client sich einloggt, prüft der Server die Accountdaten, Namen und Passwort. Danach sendet der Server die Liste der Spielwelten. Die anfänglichen Server werden wohl nur eine Spielwelt haben und daher eine feste Liste senden. Wenn der Client einen Charakter erzeugt, muss ggf. eine neue Welt erschaffen werden. Sobald der Client einen Charakter startet, muss ggf. der erste Dungeonlevel erzeugt werden und der Charakter in die Spielwelt eingesetzt werden. D.h. es muss ein freier Platz im Dungeon gefunden werden. Dadurch verändert sich der Dungeonlevel und allen anderen Clients im selben Dungeonlevel wird eine

Aktualisierung gesendet. Wenn der Client lange nichts tut, testet der Server die Verbindung mit einer Alive Nachricht. Der Server verarbeitet Aktionen der Clients und Monster und sendet ggf. Aktualisierungen an alle betroffenen Clients. Wenn ein Client sich ausloggt, wird die Figur aus dem Dungeon entfernt. Ein Prototyp-Server kann die Clientdaten danach vergessen. Ein ausgebauter Server wird beim nächsten Einloggen, den Client wieder mit allen Sachen wiederherstellen. Es ist den Gruppen überlassen, ob ein Charakter nach dem Aktivieren an der selben Stelle oder in Dungeonlevel 1 auftaucht.

1.4 Wer schickt was - Namenspräfixe

Jede Klasse wird entweder vom Client oder vom Server gesendet. Wenn eine Klasse mit dem Präfix **DMS** beginnt, handelt es sich um eine Nachricht, die der Server an den Client sendet. Die Client Nachrichten an den Server beginnen mit dem Präfix **DMC**.

2 Nachrichtenfolgen

Jede Java Klasse enthält Kommentare, die unbedingt gelesen werden sollten.

Viele Client Anfragen haben eine korrespondierende Server Nachricht. Wie zum Beispiel **DMCLogin** und **DMSLoginResult**. Die Result Nachrichten enthalten meist ein **boolean Ok** und einen **String Reason**. Wenn **Ok** auf **true** steht, war die Anfrage erfolgreich, sonst nicht. Der Server sollte in **Reason** einen Grund bzw. eine Fehlermeldung mitliefern.

2.1 Einloggen

Die erste Nachricht an den Server ist **DMCLogin**. Danach wartet der Client auf die Antwort in Form von **DMSLoginResult**. Wenn das Login erfolgreich war sendet der Server die Liste der Welten namens **DMSWorldList**. Der Client kann diese anzeigen und darauf warten, dass der Benutzer sich eine aussucht oder der Client sucht sich automatisch eine aus. Ein ausgebauter Server schickt die **DMSWorldList** auch, wenn sie sich verändert, weil z.B. eine Welt hinzugekommen ist. Ob ein Account von mehreren Clients gleichzeitig benutzt werden kann, ist wahlfrei.

2.2 Ausloggen

Um auszuloggen sendet der Client eine **DMSLogout**. Aufgrund der Trägheit des Netzwerks, muss der Client damit rechnen, dass er vom Server noch weitere Spielnachrichten bekommt, bevor er schließlich die Antwort **DMSLogoutResult** erhält.

2.3 Spiel beginnen - Charakter aktivieren

Um endlich zu spielen, muss der Client eingeloggt sein und sendet eine **DMCActivateCharacter**. Der Server antwortet mit einer **DMCActivateCharacterResult**. Der Server wird wahrscheinlich nur einen aktiven Character pro Client erlauben. Danach wird der Server schon bald eine **DMSGameState** senden.

2.4 Spiel beenden - Charakter deaktivieren

Wenn man genug hat, sendet der Client eine **DMCDeactivateCharacter** woraufhin der Server nach geraumer Zeit mit einer **DMCDeactivateCharacterResult** antworten wird.

2.5 Aktionen

Der Spieler kann Aktionen durchführen, wie beispielsweise ein Feld nach links gehen. Dafür sendet der Client eine **DMCAction**. Der Server reagiert nach eigenem Ermessen.

2.6 Neuen Charakter erzeugen, alten löschen

Während ein Client eingeloggt ist aber, noch nicht spielt, kann er neue Charaktere erzeugen (**DMCCreateCharacter**) oder alte löschen (**DMCDeleteCharacter**). Der Server antwortet darauf mit **DMSCreateCharacterResult** bzw. **DMSDeleteCharacterResult** und ggf. einer **DMSWorldList** Nachricht.

2.7 Fehler

Wenn eine Nachricht unerwartet kam, oder fehlerhafte Daten enthält, kann der Client eine **DMCProtocolError** und der Server eine **DMSProtocolError** senden. Der Empfänger kann die Nachricht in seinen Logdateien speichern, was das Debugging erheblich erleichtert.

3 Dungeon-Level

Ein Dungeonlevel besteht aus einem zweidimensionalen Array aus Feldern. Auf jedem Feld kann höchstens ein Wesen (**DMSMovable**) stehen und ein Anzahl von Dingen **DMSItem** liegen. Ein Feld kann begehbar sein, wie zum Beispiel Räume oder offene Türen. Es kann sichtbar (**Visible**) sein, d.h. durch Licht erhellt, oder bekannt (**Known**), also schon mal gesehen. Die Attribute **Visible** und **Known** sind optional zu implementieren. Auf einem Feld kann außerdem ein Hindernis (**Obstacle**) stehen, wie eine Tür. Zum Schluß gibt es noch ein optionales Stilattribut (**Style**).

3.1 Wesen, Monster, Spieler

Jedes Wesen hat einen eindeutigen und identifizierenden Namen (**UniqueName**) und einen Anzeigenamen (**Name**). Der eindeutige Name ist eine vom Server vergebene eindeutige ID. Zum Beispiel wird eine Gruppe Riesenratten immer den Name *Giant Rat* tragen aber die **UniqueName** Variablen können *R1* bis *R5* heißen. Jedes Wesen kann Zusatzattribute **Attributes** besitzen. Das Format ist frei. Ein Vorschlag ist die Form 'Name=Value'. Also zum Beispiel 'type=self' oder 'aura=blue'. Manche Wesen, wie Charaktere, können eigene Dinge besitzen. Wesen haben ein **Level** und **Stats**. Ein Beispiel für Stats ist "Strength" hat den Wert 10.0.

3.2 Dinge, Sachen und Gegenstände

Jedes Ding hat genau wie die Wesen einen eindeutigen Namen (**UniqueName**) und einen Anzeigenamen (**Name**). Ein Ding kann auf dem Boden liegen, dann hat es eine Koordinate (**X,Y**). Es kann Attribute haben, mit einem Format wie bei Wesen beschrieben. Und es hat einen Typ (**Type**).

3.3 Chat

Texte, Chat, Log, Statusmeldungen legt der Server im Vector Chat ab. Diese haben das Format "Typ: Text", also beispielsweise "chat: Igor shouts Help me!", oder "combat: you hit the giant rat for 3 hits".

3.4 Ganzes Level und Teillevel

Da ein Level eine grosse Menge Daten sind, schickt der ausgebaute Server die **DMSGGameState** Nachricht nur, wenn sich viel geändert hat. Ansonsten sendet der Server **DMSGGameChange** Nachrichten. Die **DMSGGameChange** besteht aus einer Liste aus Änderungen (**Changes**), die von 0 bis `size()-1` durchgearbeitet werden muss.

3.5 Aktionen

Aktionen bestehen immer aus einem Namen und 0 bis n Parametern. Die Basisaktionen sind im Kommentar der **DMCAction** Klasse beschrieben. Aktionen können eine bestimmte Dauer haben. Zum Beispiel macht der Client einen Schritt auf ein anderes Feld und muss danach eine halbe Sekunde warten, bis er wieder etwas anderes unternehmen kann. Dadurch haben mehrere Spieler die gleichen Chancen unabhängig vom Durchsatz der Netzwerkverbindung. Der Prototyp muss dies noch nicht beherrschen. Überdies sollte der Server Aktionen, die während der Wartezeit kommen, nicht einfach verwerfen, sondern sich merken und eine davon nach der Wartezeit automatisch ausführen. Das reduziert die Verzögerungen durch Latenz und Lag, das von Online-Spielern gefürchtet ist (siehe unten). Auch dies muss der Prototyp noch nicht beherrschen.

4 Links

<http://de.wikipedia.org/wiki/Lag>