Nico Bachmann, Vladimir Buser, Maria Desteffani, Andrina Geller

Netzwerkprotokoll

Allgemein

Der allgemeine Aufbau unserer Protokoll Befehle ist:

BEFEHL--<[parameter1]>-<[parameter2]>

Einige Befehle benötigen keine Parameter, andere sogar mehrere. Der eigentliche Befehl und die Parameter sind voneinander mit - getrennt.

Beispiel:

```
QUIT--
CHAT--hello!
WHISPER--RUE-hi
```

Die In/Out Instanzen welche die Nachrichten versenden prüfen nur ob der Befehl existiert und ggf. eine Nachricht dabei ist. Wie die Nachricht weiter verarbeitet wird, müssen die entsprechenden Klassen selbst definieren.

Bsp: **`WHISPER--`** definiert die Empfängerklasse (Chat). Diese muss dann **`RUE-hi`** weiter verarbeiten.

utility.IO - Nachrichten an Client

Das **`utility.IO`** Package stellt verschiedene Klassen zur Verfügung um Nachrichten zu Senden und Empfangen.

Um Nachrichten vom Server an den Client(s) zu senden:

```
import utility.IO.*

class MyClassCanSendToClients {

      SendToClient sendToClient = new SendToClient();

      public void s˙endHelloWorld() {

            // Send to one Client
            // sendToClient.send(ClientHandler, Command, Message)
            sendToClient.send( recipient, CommandsToClient.PRINT, "Hello World" );
            // prints Hello World on client console

            // Send to everyone on the Server
            sendToClient.serverBroadcast( CommandsToClient.PRINT, "Hello World" );
            // prints Hello World on all client consoles

      }

}
```

Zusätzlich muss natürlich der ClientHandler des Empfängers bekannt sein damit.

Es muss lediglich **`utility.IO.*`** importiert werden und ein neues **`SendToClient`** Objekt erstellt werden. Anschliessend kann über dieses Objekt gesendet und empfangen werden.

utility.IO - Nachrichten an Server

Sehr ähnlich gestaltet sich das Senden von Nachrichten an den Server:

```
import utility.IO.*

class MyClassCanSendToServer {

      SendToServer sendToServer = new SendToServer();

      public void sendHelloWorld() {

            // Send a message to the Server
            // sendToServer.send(Command, Message)
            sendToServer.send(CommandsToServer.PRINT, "Hello World" );
            // prints hello world to the server console

      }

}
```

Beim Senden an den Server braucht es keinen "Empfänger" da es nur einen Server gibt.

CommandsToServer und **CommandsToClient** sind die beiden *Enums* mit den definierten Befehlen.
Da man zum Versenden von Nachrichten einen dieser Commands eingeben muss, ist es unmöglich dass ein unbekannter Command versendet wird.

Nico Bachmann, Vladimir Buser, Maria Desteffani, Andrina Geller

utility.IO - Empfangen von Nachrichten

Nachrichten kommen alle an einem Zentralen Ort an. Auf der Serverseite beim **`ClientHandler`** bzw. dem **`ClientHandlerIn-Thread`** und auf Clientseite beim **`ClientIn-Thread`**. Diese warten permanent auf neue Nachrichten und geben sie direkt weiter an **`ServerReceive`** bzw. **`ClientReceive`**. Dort wird die Nachricht dann decodiert, validiert und an den richtigen Ort weitergeleitet.

Wie kommt nun eine Nachricht von Client-/Server-Receive an die richtige Adresse (zb. Gamelogic Klassen) ?
Die Klassen welche eine Nachricht erwarten, definieren eine Variable welche sie kontinuierlich prüfen. Der Client-/Server-Receive kann dann über einen Setter diese Variable verändern und so der Klasse Informationen übergeben.

Weil die Empfänger-Klassen nie direkt den Traffic lesen sondern bloss eine lokale Variable entscheidet alleine der Client-/Server Receive welche Nachrichten wohin gesendet werden. So ist auch sichergestellt, dass keine ungewollten Nachrichten am falschen Ort landen.

Achtung:

Die receive Methode blockiert solange, bis eine Nachricht eingetroffen ist. Der Code läuft in dieser Zeit nicht weiter! Wenn also im Hintergrund etwas empfangen werden soll, während etwas anderes weiterläuft, sollte das Receive in einen Thread verlegt werden.

Die Client-/Server-Receive Klassen dürfen nur diese ReceiveFromProtocol.setMessage(msg) methode ausführen, da ansonsten auch diese Klassen blockieren oder den Traffic drastisch verlangsamen!

utility.IO - ReceiveFromProtocol

Das ganze wird über eine Instanz der **`ReceiveFromProtocol`**-Klasse implementiert:

```
import utility.IO.*

class MyClassCanReceiveMessages {

      ReceiveFromProtocol receiveFromProtocol = new ReceiveFromProtocol();

      public void printReceivedMessages {

            String message = receiveFromProtocol.receive(); // Blocks and waits for message

            System.out.println(message);

      }

}
```

Anschliessend muss diese Instanz im Client-/Server-Receive angegeben werden, damit eine Nachricht auch dort ankommt:

```
case COMMAND:

MyClassCanReceiveMessages.receiveFromProtocol.setMessage(msg); //


break;
```

Falls ein neuer **`Command`** benötigt wird, muss dieser auch noch im **`CommandsToClient`** bzw **`CommandsToServer`** hinzugefügt werden.

Unser Netzwerkprotokoll ist in ein Server- und ein Client-Protokoll unterteilt. Der Client sendet dabei seine Eingabe (Spalte “EINGABE CLIENT”), welche im Client-Protokoll umgewandelt wird zum eigentlichen Befehl (Spalte “BEFEHL CLIENT”). Der Client sendet somit die Befehle im blauen Feld an den Server, wenn nötig antwortet dieser auch mit einem “Befehl” bzw. Codewort in gelber Farbe, damit der Client die Antwort des Servers als zu seinem Befehl gehörig erkennt. Commands die das GUI betreffen besitzen teilweise nur einen Command vom Server an den Client und nicht andersherum.

Noch sind nicht alle Befehle im Code implementiert, sobald sie relevant werden und auch etwas Sinnvolles tun können, werden sie im Code hinzugefügt (z.B. Karten ziehen). Was noch nicht implementiert ist, ist in grau markiert.

Nico Bachmann, Vladimir Buser, Maria Desteffani, Andrina Geller

Chat

FUNKTION	BEFEHL CLIENT	EINGABE CLIENT	BEFEHL SERVER	NACHRICHT AN CLIENT	NACHRICHT AN ANDERE CLIENTS	KOMMENTARE
Client schickt Nachricht an alle anderen Clients	CHAT--<message>	/chat <message>	CHAT--<message>	<client>: <message>	<client>: <message>	
Client schickt Nachricht an alle Clients in der selben Lobby	LOBBYCHAT--<message>	/lobbychat <message>		[lobby] <client>: <message>	<client> [lobby]: <message>	
Client schickt Nachricht an einen bestimmten Client	WHISPER--<otherClient>-<message>	/whisper <otherClient> <message>		<client> to <otherClient>: <message>	<client> to <otherClient>: <message>	Nur der Absender und der Empfänger bekommen eine Nachricht. “Nachricht an andere Clients” meint hier also nicht alle Spieler.

Spielereigenschaften

FUNKTION	BEFEHL CLIENT	EINGABE CLIENT	BEFEHL SERVER	NACHRICHT AN CLIENT	NACHRICHT AN ANDERE CLIENTS	KOMMENTARE
Client wechselt Username	CHANGENAME--<newName>	/nick <newName>		SUCCESS! You're now called: <newName> *oder* SORRY! This tribute already exists. Please try another name."	<client> is now called <newName>.	
Client wählt einen Charakter aus	SELECTCHAR--<charNr>	/char <charNr>		-	-	Die Spielcharaktere unterscheiden sich nur optisch.
Client lässt sich seine ID ausgeben	ID--	/id		Your ID is <id>.	-	Die ID wird später für Reconnects benötigt. Der Befehl ist nützlich, falls ein Spieler seine ID nicht (mehr) weiss.

Gameplay

FUNKTION	BEFEHL CLIENT	EINGABE CLIENT	BEFEHL SERVER	NACHRICHT AN CLIENT	NACHRICHT AN ANDERE CLIENTS	KOMMENTARE
Client gibt Bescheid, dass er bereit für das Spiel ist	READY--	/ready		You are now waiting...	<client> is ready.	
Client gibt Bescheid, dass er nicht mehr für das Spiel bereit ist	UNREADY--	/unready		You are not waiting anymore.	<client> is not ready.	
Client fragt nach aktuellem Spielstand	STATUS--			-	-	
Client würfelt	ROLLDICE--<augenzahl>	/rolldice		You rolled <number>	<client> rolled <number>	
Client würfelt 4er-Würfel	ROLLDICE--<augenzahl>	/dicedice		You rolled <number>	<client> rolled <number>	
Client zieht eine Karte	GETCARD--<cardNr>	/card <cardNr>		You got the <cardName>	<client> got the <cardName> card.	

von mehreren				card.		
Client tauscht Karte mit anderem Client	SWITCHCARDS--<otherClient>-<ownCard>-<otherCard>	/switch <otherClien> <ownCard> <otherCard>		You switched <ownCard> with <otherClient>'s <otherCard>	<client> and <otherClient> switched cards. *oder* <client> switched their <ownCard> with your <otherCard>.	Bei "Nachricht an alle Clients" geht die erste Nachricht an denjenigen, mit dem Karten getauscht wird. Die zweite Nachricht geht an alle anderen.
Client bewegt Spielfigur	MOVECHAR--<destination>	/move <destination>		-	-	
Client wählt eine Antwort bei einer Quiz-Frage	QUIZ--<answer>	/quiz <answer>		<client> 's answer: <answer> is correct. <client> 's answer is wrong.	<client> 's answer: <answer> is correct. <client> 's answer is wrong.	Bei Quizfragen, die alle bekommen, wird nicht mitgeteilt, wer welche Antwort gegeben hat. Bekommt nur einer die Quizfrage, könnten andere die Antwort auch sehen (noch abzuklären)
Client wählt beim Würfeln sein Zielfeld aus	WWCD--<Zielfeld>	/winnerwinnerchickendinner <Zielfeld>		<client> has rich parents. <client> moved from: <alte Position> to <Zielfeld>	<client> has rich parents. <client> moved from: <alte Position> to <Zielfeld>	

GUI

Die Befehle vom GUI gehen nur in eine Richtung,

FUNKTION	BEFEHL CLIENT	EINGABE CLIENT	BEFEHL SERVER	NACHRICHT AN CLIENT	NACHRICHT AN ANDERE CLIENTS	KOMMENTARE
der Text auf dem GUI-Startbildschirm ändert sich			PRINTGUISTART--			Dieser Befehl kann nicht vom Client aufgerufen werden, sondern wird direkt von Methoden wie z.B. askName aufgerufen, damit sich der Text auf dem Bildschirm entsprechend den System.out.println() in der Konsole anpasst.

Lobbies

FUNKTION	BEFEHL CLIENT	EINGABE CLIENT	BEFEHL SERVER	NACHRICHT AN CLIENT	NACHRICHT AN ANDERE CLIENTS	KOMMENTARE
Druckt eine Liste von allen Spielern im Game aus	PRINTUSERLIST--	/printUserList>	PRINT--	String mit allen Usern		
Druckt alle existierenden Lobbies unabhängig vom Status	PRINTLOBBIES--	/printLobbies	PRINT--	String mit allen Lobbies		
Druckt die Lounging List aus: also alle Lobbies mit dem jeweiligen Status und ihre Nutzer..	PRINTLOUNGINGLIST--	/printLoungingList	PRINT--	String mit allen Lobbies und ihrem Status und zusätzlich noch alle Spieler in der jeweiligen Lobby.		
Druckt alle offenen Lobbies	PRINTOPENLOBBIES--	/printOpenLobbies	PRINT--	String mit allen offenen Lobbies		

Druckt alle fertigen Lobbies	PRINTFINISHEDLOBBIES--	/printFinishedLobbies	PRINT--	String mit allen fertigen Lobbies		
Druckt alle laufenden Lobbies	PRINTONGOINGLOBBIES--	/printOnGoingLobbies	PRINT--	String mit allen laufenden Lobbies		
Spricht mit der Lobby Klasse	LOBBY--	/lobby				
Lobby erstellen	CREATELOBBY--	/createLobyb	PRINT--	String mit antwort ob Lobby erstellt wurde oder nicht		
Lobby wechseln	CHANGELOBBY--	/changeLobby	PRINT--	message if changing the lobby was succesful		
Gibt alle Players in der Lobby zurück	PRINTPLAYERSINLOBBY--	/printPlayersInLobby	PRINT--	String mit allen Spielern in der Lobby		Es handelt sich um die Lobby von dem Spieler, der den Command eingegeben hat.
Client verlässt das Spiel	QUIT--	/quit		-	<client> left the game.	