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> oder per GUI	CHAT--<message>	<client>: <message> und grafische Anzeige im GUI	<client>: <message> und grafische Anzeige im GUI	
Client schickt Nachricht an alle Clients in der selben Lobby	LOBBYCHAT--<message>	/lobbychat <message> oder per GUI		[lobby] <client>: <message> und grafische Anzeige im GUI	<client> [lobby]: <message> und grafische Anzeige im GUI	Im GUI kann zwischen Lobbychat und globalem Chat mit Hilfe von Buttons gewechselt werden
Client schickt Nachricht an einen bestimmten Client	WHISPER--<otherClient>-<message>	/whisper <otherClient> <message> oder per GUI (Chat-Fenster)		<client> to <otherClient>: <message> und grafische Anzeige im GUI	<client> to <otherClient>: <message> und grafische Anzeige im GUI für den Empfänger	Nur der Absender und der Empfänger bekommen eine Nachricht. “Nachricht an andere Clients“ meint hier also nicht alle Spieler. Im GUI kann der Befehl im Chat eingegeben werden.

Spielereigenschaften

FUNKTION	BEFEHL CLIENT	EINGABE CLIENT	BEFEHL SERVER	NACHRICHT AN CLIENT	NACHRICHT AN ANDERE CLIENTS	KOMMENTARE
Client wechselt Username	CHANGENAME--<new Name>	/nick <newName> oder per GUI		SUCCESS! You're now called: <newName> *oder* SORRY! This tribute already exists. Please try another name." und im GUI ändert sich das Namelabel im Menu und Game		Duplikate oder unerlaubte Namen werden vom System automatisch abgeändert.
Der Client kann so seinen Charakter ändern.	CHANGECHARACTER –<newCharacterNum ber>	GUI	ENABLECHAR GUI–<oldChar acterNumber >> DISABLECHA RGUI–<newCh aracterNumb er>	Im Gui wird der alte Charakter wieder freigegeben und der gewählte disabled.	Im Gui wird der alte Charakter wieder freigegeben und der gewählte disabled.	Charakter welche disabled sind, sind nicht wählbar.

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 oder per GUI		You are now waiting... (GUI und Konsole)	<client> is ready. und grafisch im GUI (Game-Tracker))	
Client gibt Bescheid, dass er nicht mehr für das Spiel bereit ist	UNREADY--	/unready		You are not waiting anymore. (GUI und Konsole)	<client> is not ready. und grafisch im GUI (Game-Tracker)	
Client würfelt	ROLLDICE	/rolldice oder per GUI		You rolled <number> und grafisch im GUI (Game-Tracker)	<client> rolled <number> und grafisch im GUI (Game-Tracker)	
Startet das Spiel	START--	/start oder per GUI				Kann erst aufgerufen werden, wenn min. 2 Spieler ready sind.
Spielfigur bewegt sich über			GUIMOVECHARACTER-<tokenNum	Das entsprechende Player-Token	Das entsprechende Player-Token bewegt sich über das Brett.	Geschieht automatisch nach dem Würfeln.

das Brett			ber>--<newFieldNumber>	bewegt sich über das Brett.		
Client wählt eine Antwort bei einer Quiz-Frage	QUIZ--<answer>	/quiz <answer> oder per GUI		<client> 's answer: <answer> is correct. <client> 's answer is wrong. und grafisch im GUI (Game-Tracker)	<client> 's answer: <answer> is correct. <client> 's answer is wrong. und grafisch im GUI (Game-Tracker)	
Client wählt beim Würfeln sein Zielfeld aus	WWCD--<Zielfeld>	/winnerwinnerchickendinner <Zielfeld>		<client> has rich parents. <client> moved from: <alte Position> to <Zielfeld> und grafisch im GUI (Game-Tracker)	<client> has rich parents. <client> moved from: <alte Position> to <Zielfeld> und grafisch im GUI (Game-Tracker)	Eingabe geht im GUI per Chat.
Würfelt einen 4-er Würfel	DICEDICE--	/dicedice oder per GUI		Einer der drei 4-er Würfel wird im GUI ausgegraut und disabled		Der 4er-Würfel lässt sich nur 3 mal benutzen pro Spiel und Player.
Sagt dem Client wie viele spezielle			DICEDICELEFT--			Wird als "Hilfscommand" benutzt für das Würfeln mit dem

Würfel er/sie übrig hat.						4-er Würfel

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--	Text im GUI ändert sich		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.
Stellt den Charakter in der Charakterauswahl auf besetzt.	DISABLECHARGUI–<characterNumber>	per GUI automatisch nach Charakterauswahl	DISABLECHARGUI–<characterNumber>	Der besagte Charakter wird im GUI disabled.	Die anderen Clients können im GUI sehen, welche Charakter nun noch zur Verfügung stehen und welche nicht.	

Stellt den Charakter in der Charakterauswahl frei.	ENABLECHARGUI–<characterNumber>	per GUI automatisch nach Charakterauswahl	ENABLECHARGUI--<characterNumber>	Der besagte Charakter wird im GUI freigegeben.	Die anderen Clients können im GUI sehen, welche Charakter nun noch zur Verfügung stehen und welche nicht.	
Setzt das richtige Bild des Charakters in das Spiel unten rechts.	SETCHARTOKEN–<tokenNumber>--<characterNumber>	per Gui automatisch nach dem Drücken des Ready-Buttons	SETCHARTOKEN–<tokenNumber>--<characterNumber>	Der ausgewählte Charakter wird im richtigen Token angezeigt.	Der ausgewählte Charakter wird im richtigen Token angezeigt.	
Setzt alle Charakter richtig. (Ist vor allem wichtig für den Spectator Mode)	SETALLCHARTOKEN--	geschieht automatisch beim Wechseln in den Game-Screen	SETALLCHARTOKEN--	alle Player Tokens im Game-Screen werden richtig gesetzt.		Ist nützlich, wenn ein Client erst einer Lobby beitritt, wenn einige Spieler schon ready sind.
Setzt alle Charakter richtig ein.	SETALLCHARS--	Geschieht automatisch nach dem Beitritt in eine Lobby.	ENABLECHARGUI–<characterNumber> DISABLECHARGUI–<characterNumber>	Im Character Selection Screen werden alle Charaktere darauf geprüft, ob sie vergeben sind, oder nicht und entsprechend en-/disabled.		

Gibt den Charakter vom eigenen User frei, damit der Charakter wieder frei ist, wenn man die Lobby wechselt.	ENABLECURRENTCHARGUI--	Geschieht automatisch bei LobbyWechsel.	ENABLECHARGUI-<characterNumber>		Die Spieler in der vorherigen Lobby des Spieler bekommen den Charakter wieder frei zur Auswahl.	Wird benutzt, um beim Lobby-Wechsel den richtigen Charakter wieder freizugeben.
Printed in das kleine Fenster unterhalb vom Chat, was gerade passiert im Spiel.		Geschieht automatisch während dem Spiel.	PRINTGUIGAMETRACKER--	Der Client kriegt über den Broadcast Bescheid, was im Spiel gerade passiert.	Die anderen Clients kriegen alle die genau gleiche Nachricht, weil es ein Broadcast ist.	
Druckt alle Gewinner und Gewinnerinnen im Highscore aus.		Wird bei Änderung automatisch aktualisiert oder per Button im GUI refreshed.	PRINTWINNERSGUI--	Der Client kann in seinem Highscore die ausgedruckte Highscore Liste sehen.		Der Highscore ist persistent und druckt die 10 besten Spieler und Spielerinnen aus.
Druckt die LoungingList Menu aus.		Wird bei Änderungen automatisch aktualisiert oder im GUI per Button refreshed.	PRINTFRIENDSGUI--	Der Client kann in seinem Menu die ausgedruckte LoungingList sehen.	-	
Druckt alle Lobbies im Menu aus.		Wird bei Änderungen automatisch	PRINTLOBBIESGUI--	Der Client kann bei sich im Menu die	-	

		aktualisiert oder im GUI per Button refreshed.		ausgedruckte LobbyList sehen.		
Setzt das Label auf den Namen des Clients.		Wenn der SPieler oder die Spielerin seinen/ihren Name ändert wird das Label automatisc aktualisiert.	NAME--	Im GUI des Clients ändert sich das Label mit dem Namen des Clients.	-	
Markiert den Spieler oder die Spielerin mit einem orangen Kreis um den Charakter.		Geschieht automatisch, wenn der Spieler an der Reihe ist.	MARKPLAYER--	Der Char des Clients wird orange umrandet.	-	
Benachrichtigt den richtigen Client, dass er/sie eine Quizfrage zu beantworten hat, indem die Quiz-Buttons freigegeben werden.		Geschieht automatisch, wenn der Spieler an der Reihe ist.	YOURQUIZ--	Im Gametracker wird der Name des Clients, welcher an der Reihe ist, ausgedruckt. "Quizfrage an:" + Client xy.	Im Gametracker wird der Name des Clients, welcher an der Reihe ist, ausgedruckt. "Quizfrage an:" + Client xy.	
Benachritigt den richtigen Client, dass er/sie an der Reihe ist.		Geschieht automatisch, wenn der Spieler an der Reihe ist.	YOURTURN--	Im Gametracker wird wird eine Nachricht aus an alle geschickt,	Im Gametracker wird wird eine Nachricht aus an alle geschickt, welcher Spieler	

				welcher Spieler oder Spielerin an der Reihe ist.	oder Spielerin an der Reihe ist.	
Prüft ob ein Charakter schon vergeben ist oder nicht.	CHECKIFCHARSTAKEN--<characterNumber>		ENABLECHAR–<characterNumber> oder DISABLECHAR–<characterNumber>	Im Character Selection screen wird der entsprechende Charakter en-/disabled.		
Überprüft, welche Charakter in der Lobby schon vergeben sind.	CHECKALLCHARS--	Geschieht automatisch bei Lobbybeitritt	ENABLECHAR–<characterNumber> oder DISABLECHAR–<characterNumber>	Im Character Selection Screen werden alle Charaktere überprüft und entsprechend en-/disabled		

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--	/createLobby oder per GUI	PRINT--	String mit antwort ob Lobby erstellt wurde oder nicht		
Lobby wechseln	CHANGELOBBY–	/changeLobby oder per GUI	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 oder per GUI		-	<client> left the game.	

High score

FUNKTION	BEFEHL CLIENT	EINGABE CLIENT	BEFEHL SERVER	NACHRICHT AN CLIENT	NACHRICHT AN ANDERE CLIENTS	KOMMENTARE
Gibt die Bestenliste zurück	PRINTHIGHSCORE–	/printHighscore oder automatisch im GUI (bzw. per refresh button)	PRINTHIGHSCORE--	String mit den besten zehn Ergebnissen oder “empty High Score“, falls noch kein Eintrag vorhanden ist.		

Musik

FUNKTION	BEFEHL CLIENT	EINGABE CLIENT	BEFEHL SERVER	NACHRICHT AN CLIENT	NACHRICHT AN ANDERE CLIENTS	KOMMENTARE
Spiel Musik ab.	MUSIC--	Geschieht automatisch durch Änderungen im GUI (z.B. Start Button drücken)	MUSIC--	Musik ertönt.	Musik ertönt teilweise auch (z.B. Start-Button drücken).	