**Die Client/Server Architektur**

Zur Übertragung von sämtlichen Informationen unseres Spiels verwenden wir eine Client/Server Architektur. Spieler 1 übernimmt die Rolle des Servers und baut eine Verbindung zum Spieler 2 (Client) auf, um Daten zu senden. Dieser Spieler baut ebenfalls eine erneute Verbindung zu der bestehenden mit Spieler 1 auf. Auf dem zuerst beschriebenen Kanal werden Informationen wie x-Koordinaten des Schlägers, die x, y Koordinaten des Balls und der Punktestand jedes Spielers, an Spieler 2 gesendet. Auf dem zweiten Kanal erfolgt lediglich die Sendung der x-Koordinate von Spieler 2 an Spieler 1.

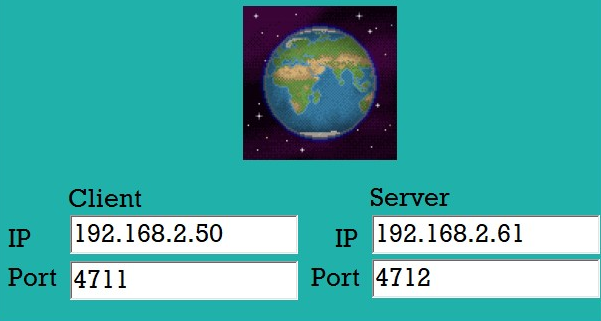
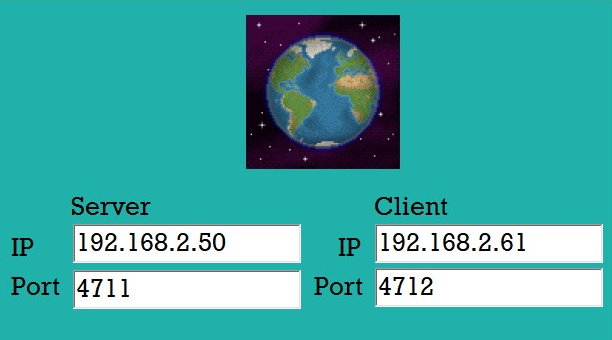
 Server Client



Client Server

Spieler 1 (Server) Spieler 2(Client)

Mithilfe der IP-Adresse und eines Ports können diese Daten-Pakete zu dem anderen Rechner gesendet werden. Wie man in den Abbildungen sehen kann, startet der Server seinen Dienst. In diesem Fallbeispiel hat der Server die IP-Adresse 192.168.2.61 und den Port 4712. Der Client verbindet sich dann mit dem Server, hiermit wäre Datenkanal 1 hergestellt und der Server schickt dem Client seine Daten. Um Datenkanal 2 herzustellen startet der Client seinen Dienst mit der IP-Adresse 192.168.2.50 und dem Port 4711 mit dem sich der Server dann verbindet.



Serverform Clientform

Datenkanal 2 Datenkanal 1

Wir benutzen das TCP-Protokoll, welches den Vorteil hat, dass die Daten, die wir senden 1:1 so ankommen wie sie gesendet wurden und kleine Verzögerungen, die in unserem Fall kaum vorhanden sind. Eine viel wichtigere Rolle ist der Spielstand jedes Spieler, der vom Server gesendet wird, dieser sollte immer ohne Verluste beim Client ankommen, daher ist dieses Protokoll für uns die richtige Wahl. Falls beim TCP-Protokoll beim Senden Daten verloren gehen, sorgt es dafür, dass diese noch einmal übertragen werden.

**Der Prozess des Servers beim Verbindungsaufbau**

Beim Starten des Serverdienstes wird durch die IP-Adresse und der Portnummer ein Kommunikationsendpunkt bestimmt. Nach Erstellung einer Socket-Instanz muss man noch Kommunikationsendpunkt und Socket zusammenführen. Um diesen Socket von einem passiven in einen aktiven Zustand zu bringen, versetzen wir den Socket in den Listening Zustand. Nach der Ausführung der Accept Methode im ClientHandler (Abbildung 2.1) blockiert der Server die Server-Anwendung solange sich kein Client verbindet.

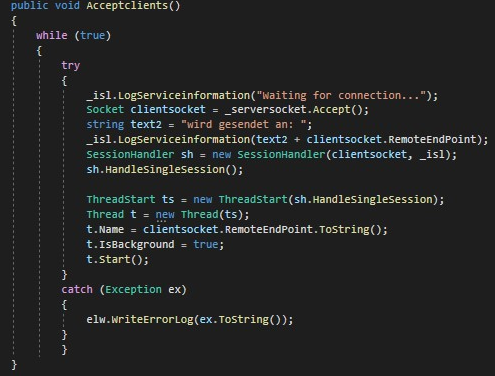


Abbildung 2.1

In der Klasse Sessionhandler (Abbildung 2.2) wird der eingehende Request eingelesen und an den RequestHandler(Abbildung 2.3) weitergeleitet, der den Request dann genauer analysiert und die wichtigen ausselektierten Informationen durch den IServiceLogger an die Anwendung selbst weiterleitet. Die Anwendungsebene nimmt diese Informationen und steuert dadurch dann den Schläger des Gegenspielers. Die lokalen Informationen werden beim Server von einem Timer (Tickrate 1ms) nach diversen Feinkalkulationen der Frequenzsteuerung (Verweis Seite) und der Tonsteuerung (Verweis Seite) an den anderen Endpunkt gesendet.



Abbildung 2.2

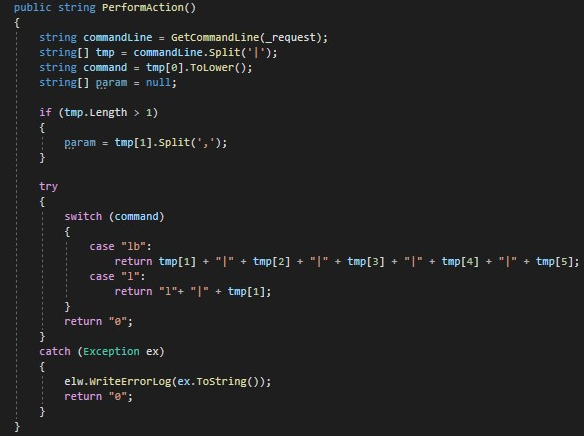


Abbildung 2.3

**Der Prozess des Clients beim Verbindungsaufbau**

Der Client hat ebenfalls eine IP-Adresse und einen Port, welche er an einen Socket binden muss. Dieser Socket verbindet sich auf Kommando mit dem anderen Kommunikationsendpunkt, der zur gleichen Zeit auf die Verbindung wartet. Nachdem die Verbindung hergestellt wurde erfolgt der gleiche Prozess wie beim Server.

**MOVE-Protocol**

­­Inhalt

[1.0 Introduction 2](#_Toc2769470)

[1.1 Purpose 2](#_Toc2769471)

[1.2 Procedure 2](#_Toc2769472)

[1.3 Structure 2](#_Toc2769473)

[1.4 Example Request 2](#_Toc2769474)

[1.5 Digits 2](#_Toc2769475)

# Introduction

# Purpose

The Move Protocol is a TCP/IP communication protocol that uses port 4711 and 4712 which is located on application layer. It is used to send information (state values) to perform our game. The protocol works with two commands that includes different parameters.

Command 1:(ball location(x), ball location(y), server paddle location(x), score (client), score (server)). Command 2: (client paddle location (x)).

# Procedure

After the connection between the two clients is established, the server is sending “lb” + his parameters and the client is sending “l” + his parameter. They are exchanging their information(parameters) every 50ms. If one response does not arrive the game session will send the value “0” and therefore the panels will stuck left.

# Structure

Command 1 (Server):

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Type | :\\ | command | ­ Separator | <param1> | Separator | <param2> | Separator | <param3> | Separator | <param4> | Separator | <param5> |
| move | :\\ | lb | | |  | | |  | | |  | | |  | | |  |

Command 2 (Client):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | :\\ | command | ­ Separator | <param1> |
| move | :\\ | l | | |  |

# Example Request

**Format:** „move:\\lb|<param1>|<param2>|<param3>|<param4>|<param5>“

**Example Request:** „move:\\lb| {400}|{30}|{5}|{11}|{10}”

**Format:** „move:\\l|<param1>“

**Example Request:** „move:\\l|{51}”

# Digits

**Action lb**

<param1> [0-560] int

<param2> [10,30,50] int

<param3> [1,2,3] int

<param4> [0-12] int

<param5> [0-12] int

**Action l**

<param1> [0-560] int

<param2> [10,30,50] int

<param3> [1,2,3] int

<param4> [0-12] int

<param5> [0-12] int