Software Challenge XML-Dokumentation Blokus

Ziel dieser Dokumentation ist es, die XML-Schnittstelle der Softwarechallenge festzuhalten.

Wir freuen uns über sämtliche Verbesserungsvorschläge. Die Dokumentation kann direkt auf GitHub editiert werden, einzige Voraussetzung ist eine kostenlose Registrierung bei GitHub. Ist man angemeldet, kann man ein Dokument auswählen (ein guter Startpunkt ist die Datei index.adoc welche Verweise auf alle Sektionen der Dokumentation enthält) und dann auf den Stift oben rechts klicken. Alternativ auch gern eine E-Mail an svk@informatik.uni-kiel.de.

Einleitung

Wie in den letzten Jahren wird zur Client-Server Kommunikation ein XML-Protokoll genutzt. In diesem Dokument wird die Kommunikationsschnittstelle definiert, sodass ein komplett eigener Client geschrieben werden kann. Es wird hier nicht die vollständige Kommunikation dokumentiert bzw. definiert, dennoch alles, womit ein Client umgehen können muss, um spielfähig zu sein.

An wen richtet sich dieses Dokument?

Die Kommunikation mit dem Spielserver ist für diejenigen, die aufbauend auf dem Simpleclient programmieren, unwichtig. Dort steht bereits ein funktionierender Client bereit und es muss nur die Spiellogik entworfen werden. Nur wer einen komplett eigenen Client entwerfen will, beispielsweise um die Programmiersprache frei wählen zu können, benötigt die Definitionen.

Hinweise

Falls Sie beabsichtigen sollten, diese Kommunikationsschnittstelle zu realisieren, sei darauf hingewiesen, dass es im Verlauf des Wettbewerbes möglich ist, dass weitere, hier noch nicht aufgeführte Elemente zur Kommunikationsschnittstelle hinzugefügt werden. Um auch bei solchen Änderungen sicher zu sein, dass ihr Client fehlerfrei mit dem Server kommunizieren kann, empfehlen wir Ihnen, beim Auslesen des XML jegliche Daten zu verwerfen, die hier nicht weiter definiert sind. Die vom Institut bereitgestellten Programme (Server, Simpleclient) nutzen eine Bibliothek um Java-Objekte direkt in XML zu konvertieren und umgekehrt. Dabei werden XML-Nachrichten nicht unbedingt mit einer newline abgeschlossen.

Die XML Dokumentation behandelt ausschließlich die Kommunikation. Falls Sie Dokumentation für den Verbindungsaufbau suchen, so finden Sie diese hier.

Spiel betreten

Wenn begonnen wird mit dem Server zu kommunizieren, muss zuallererst

```
<protocol>
```

gesendet werden, um die Kommunikation zu beginnen.

Ohne Reservierungscode

Betritt ein beliebiges offenes Spiel:

```
<join gameType="swc_2021_blokus" />
```

Sollte kein Spiel offen sein, wird so ein neues erstellt. Je nachdem ob paused in server.properties true oder false ist, wird das Spiel pausiert gestartet oder nicht. Der Server antwortet darauf mit:

- ROOM_ID ID des GameRooms

```
<joined roomId="ROOM_ID" />
```

Alle administrativen Clients werden ebenfalls darüber benachrichtigt und erhalten folgende Nachricht:

- ROOM_ID ID des GameRooms

```
<joinedGameRoom roomId="ROOM_ID" existing="false" />
```

Falls bereits ein GameRoom offen war, ist dementsprechend existing true.

Mit Reservierungscode

Ist ein Reservierungscode gegeben, so kann man den durch den Code gegebenen Platz betreten.

Join mit Reservierungscode

- RC Reservierungscode

```
<joinPrepared reservationCode="RC" />
```

Welcome Message

Der Server antwortet darauf erst, wenn der zweite Client ebenfalls verbunden ist:

- ROOM_ID Id des GameRooms
- COLOR Spielerfarbe. Da bei Blokus jeder Spieler zwei Farben spielt, ist hier nur one oder two angegeben

- STATUS GameState wie in Status

```
<joined roomId="ROOM_ID" />
<room roomId="ROOM_ID">
  <data class="welcomeMessage" color="COLOR"></data>
</room>
<room roomId="ROOM_ID">
  STATUS
</room>
```

Als erstes kommt eine Bestätigung, dass das Spiel betreten wurde. Hier kann die Raum-ID festgestellt und für die weitere Kommunikation gespeichert werden. Nach der Willkommensnachricht, welche dem Spieler mitteilt, ob er erster oder zweiter Spieler ist, wird der initiale Spielstatus gesendet.

Zug-Anforderung

Eine einfache Nachricht fordert zum Zug auf:

```
<room roomId="ROOM_ID">
  <data class="sc.framework.plugins.protocol.MoveRequest" />
</room>
```

Züge senden

Der Move

Der Move ist die Antwort auf den MoveRequest des Servers.

Senden

Der Move ist der allgemeine Zug, der in verschiedenen Varianten gesendet werden kann.

`ROOM_ID`

: ID des GameRooms, aus dem `MoveRequest`.

`ZUG`

: Zug wie in ZUG

```
<room roomId="ROOM_ID">
  ZUG
</room>
```

ZUG

`COLOR`

Farbe des zu setzenden Spielsteines (`BLUE`,`YELLOW`,`RED`,`GREEN`)

`KIND`

Typ des zu setzenden Spielsteines (siehe Spielsteine)

`ROTATION`

Drehung des Spielsteines (`NONE`,`RIGHT`,`MIRROR`,`LEFT`)

`FLIPPED`

Ob der Spielstein umgedreht wurde (`true`,`false`)

`X`

X-Koordinate des zu ziehenden Spielsteines oder des Zielfeldes

`Y`

Y-Koordinate des zu ziehenden Spielsteines oder des Zielfeldes

Es gibt zwei Arten von Zuegen. Entweder man setzt einen eigenen noch nicht gesetzten Stein auf ein Zielfeld:

```
<data class="sc.plugin2021.SetMove">
  <piece color="COLOR" kind="KIND" rotation="ROTATION" isFlipped="FLIPPED">
    <position x="X" y="Y"/>
  </piece>
</data>
```

Oder man möchte nicht setzen, dann ist es erlaubt, auszusetzen:

```
<data class="sc.plugin2021.SkipMove">
  <color>COLOR</color>
</data>
```

Spielstein Transformationen

Bei `ROTATION NONE` ändern sich die Koordinaten der Teile des Spielsteins nicht. `(x, y) ⇒ (x, y)`

[pento L]

Bei `ROTATION RIGHT` wird der Spielstein um den Ursprung nach rechts gedreht. `(x, y) ⇒ (-y, x)`

[pento L RIGHT]

Bei `ROTATION MIRROR` wird der Spielstein um den Ursprung gespiegelt. `(x, y) ⇒ (-x, -y)`

[pento L MIRROR]

Bei `ROTATION LEFT` wird der Spielstein um den Ursprung nach links gedreht. `(x, y) ⇒ (y, -x)`

[pento L LEFT]

Wird zusätzlich `FLIPPED` auf `true` gesetzt, so werden die von der Rotation resultierenden Koordinaten anschließend noch auf der X-Achse invertiert. `(x, y) ⇒ (-x, y)`

[pento L LEFT FLIPPED]

Auf dem vorigen Bild ist das Resultat von `ROTATION LEFT` und `FLIPPED true` zu sehen.

Nach den Transformationen werden die Koordinaten noch normalisiert, d.h. sie werden an die obere linke Ecke angelegt.

Dafür werden sie mit dem minimalen X Wert aller Koordinaten und dem minimalen Y Wert aller Koordinaten subtrahiert. `(x, y) ⇒ (x - minX, y - minY)`

Schließlich werden die Koordinaten noch mit den `X, Y` Werten, die zum Move gehören, addiert, um die Koordinaten aller Teile des Spielsteins auf dem Board zu erhalten.

Debughints

Zügen können Debug-Informationen beigefügt werden:

```
<hint content="S" />
```

Damit sieht beispielsweise ein Zug so aus:

```
<room roomId="ROOM_ID">
  <data class="sc.plugin2021.SetMove">
    <piece color="COLOR" kind="KIND" rotation="ROTATION" isFlipped="FLIPPED">
      <position x="X" y="Y"/>
    </piece>
    <hint content="Dies ist ein Hint." />
    <hint content="noch ein Hint" />
  </data>
</room>
```

Spielsteine

Es gibt 21 verschiedene Arten Spielsteine. Alle haben im XML einen Namen. Diese sind:

- `MONO`
- `DOMINO`
- `TRIO_L`

- TRIO_I
- TETRO_O
- TETRO_T
- TETRO_I
- TETRO_L
- TETRO_Z
- PENTO_L
- PENTO_T
- PENTO_V
- PENTO_S
- PENTO_Z
- PENTO_I
- PENTO_P
- PENTO_W
- PENTO_U
- PENTO_R
- PENTO_X
- PENTO_Y

Spielstatus

Es folgt die Beschreibung des Spielstatus, der vor jeder Zugaufforderung an die Clients gesendet wird. Das Spielstatus-Tag ist dabei noch in einem *data*-Tag der Klasse *memento* gewrappt:

memento

- ROOM_ID Id des GameRooms
- STATUS Gamestate wie in Status

```
<room roomId="ROOM_ID">
  <data class="memento">
    STATUS
  </data>
</room>
```

Status

- Z aktuelle Zugzahl

- `R` aktuelle Rundenzahl
- `P` Spielstein, der in der ersten Runde gelegt werden muss, siehe Spielsteine
- `T` Team, welches beginnt (`ONE`, `TWO`)
- `board` Das Spielbrett, wie in Spielbrett definiert
- `blueShapes`, `yellowShapes`, `redShapes`, `greenShapes` Noch nicht gesetzte Spielsteine, siehe Nicht gesetzte Spielsteine
- `lastMoveMono` beschreibt, wenn eine Farbe alle Steine gelegt hat, ob der Mono-Stein als letztes gelegt wurde
- `validColors` alle Farben, die noch im Spiel sind
- `first` Name und auch Team des ersten Spielers.
- `second` Name und auch Team des zweiten Spielers.
- `lastMove` der zuletzt ausgeführte Zug, siehe Vorheriger Zug

```
<state class="state" turn="Z" round="R" startPiece="P">
  <startTeam class="team">T</startTeam>
  blueShapes
  yellowShapes
  redShapes
  greenShapes
  validColors
  first
  second
  board
  lastMove
  lastMoveMono
</state>
```

Spielbrett

- `X` X-Koordinate
- `Y` Y-Koordinate
- `CONTENT` Farbe des Spielsteins, der dieses Feld überdeckt (`BLUE`,`YELLOW`,`RED`,`GREEN`)

```
<board>
  <fields>
    <field x="17" y="0" content="BLUE"/>
    [...]
    <field x="17" y="2" content="BLUE"/>
  </fields>
  [...]
</board>
```

<board> enthaelt <field> Tags für alle Felder, die bereits belegt sind. Leere Felder kommen nicht vor. Grundsätzlich besteht das Spielbrett aber immer aus 20x20 Feldern, wobei das Feld links oben die X- und Y-Koordinate 0 hat und die positive X-Achste nach rechts und die positive Y-Achse nach unten verläuft.

Nicht gesetzte Spielsteine

```
<blueShapes>
  <shape>MONO</shape>
  <shape>DOMINO</shape>
  <shape>TRIO_L</shape>
  <shape>TRIO_I</shape>
  <shape>TETRO_O</shape>
  <shape>TETRO_T</shape>
  <shape>TETRO_I</shape>
  <shape>TETRO_L</shape>
  <shape>TETRO_Z</shape>
  <shape>PENTO_L</shape>
  <shape>PENTO_T</shape>
  <shape>PENTO_S</shape>
  <shape>PENTO_Z</shape>
  <shape>PENTO_I</shape>
  <shape>PENTO_P</shape>
  <shape>PENTO_W</shape>
  <shape>PENTO_U</shape>
  <shape>PENTO_R</shape>
  <shape>PENTO_X</shape>
  <shape>PENTO_Y</shape>
</blueShapes>
```

Die nicht gesetzten Steine werden durch <shape> Tags in einem <blueShapes>, <yellowShapes>, <redShapes> und <greenShapes> Tag dargestellt.

Letzter Stein

```
<lastMoveMono>
  <entry>
    <color>YELLOW</color>
    <boolean>true</boolean>
  </entry>
</lastMoveMono>
```

Jede Farbe, die alle Steine gesetzt hat, bekommt einen <entry> Tag. Dieser beschreibt fuer die Farbe im <color> Tag als <boolean>, ob der Mono Stein als letztes gesetzt wurde.

Farben im Spiel

```
<validColors>
  <color>BLUE</color>
  <color>YELLOW</color>
  <color>RED</color>
  <color>GREEN</color>
</validColors>
```

Alle Farben, die noch Züge durchführen können, werden durch `<color>` Tags dargestellt.

Erster Spieler

```
<first displayName="One">
  <color class="team">ONE</color>
</first>
```

Der erste Spieler wird wird mit dem Tag `<first>` beschrieben. Das Attribut "displayName" beinhaltet den Spielernamen des ersten Spielers. Der untergeordnete Tag `<color>` hält entweder den Wert `ONE` oder `TWO`. Hier ist dies `ONE`, also macht der erste Spieler den ersten Zug.

Zweiter Spieler

Dieser Tag beschreibt den zweiten Spieler. Die Struktur ist wie bei Erster Spieler.

Vorheriger Zug

```
<lastMove class="sc.plugin2021.SetMove">
  <piece color="BLUE" kind="PENTO_V" rotation="RIGHT" isFlipped="false">
    <position x="17" y="0"/>
  </piece>
</lastMove>
```

Der vorherige Zug hat die selbe Struktur wie ein ZUG, der gesendet wird, ausser dass das Tag `<lastMove>` und nicht `<data>` heisst. Der vorherige Zug wird in jedem Spielstatus angegeben, ausser vor dem ersten Zug.

Beispiel kompletter Spielstatus

Hier ist das XML eines kompletten beispielhaften Spielstatus, wie es der Computerspieler vom Server bekommt:

```
<room roomId="cb3bc426-5c70-48b9-9307-943bc328b503">
  <data class="memento">
    <state turn="70" round="18" startPiece="PENTO_L">
      <startTeam class="team">ONE</startTeam>
      <blueShapes/>
      <yellowShapes>
        <shape>MONO</shape>
      </yellowShapes>
      <redShapes>
        <shape>MONO</shape>
        <shape>DOMINO</shape>
      </redShapes>
      <greenShapes>
        <shape>MONO</shape>
        <shape>DOMINO</shape>
        <shape>TRIO_L</shape>
        <shape>TRIO_I</shape>
      </greenShapes>
      <validColors>
        <color>YELLOW</color>
        <color>RED</color>
        <color>GREEN</color>
      </validColors>
      <first displayName="">
        <color class="team">ONE</color>
      </first>
      <second displayName="">
        <color class="team">TWO</color>
      </second>
      <board>
        <field x="0" y="0" content="RED"/>
        <field x="1" y="3" content="GREEN"/>
        <field x="8" y="6" content="YELLOW"/>
        <field x="5" y="9" content="BLUE"/>
      </board>
      <lastMove class="sc.plugin2021.SetMove">
        <piece color="BLUE" kind="MONO" rotation="NONE" isFlipped="false">
          <position x="0" y="0"/>
        </piece>
      </lastMove>
      <lastMoveMono>
        <entry>
          <color>BLUE</color>
          <boolean>true</boolean>
        </entry>
      </lastMoveMono>
    </state>
  </data>
</room>
```

Spiel verlassen

Wenn ein Client den Raum verlässt, bekommen die anderen Clients eine entsprechende Meldung vom Server.

- `ROOM_ID` Id des GameRooms

```
<left roomId="ROOM_ID" />
```

Spielergebnis

Zum Spielende enthält der Spieler das Ergebnis:

- `ROOM_ID` Id des GameRooms
- `R1`, `R2` Text, der den Grund für das Spielende erklärt
- `CAUSE1`, `CAUSE2` Grund des Spielendes (`REGULAR`, `LEFT`, `RULE_VIOLATION`, `SOFT_TIMEOUT`, `HARD_TIMEOUT`)
- `WP1`, `WP2` Siegpunkte der jeweiligen Spieler, 0 verloren, 1 unentschieden, 2 gewonnen
- `S1`, `S2` Punkte des jeweiligen Spielers
- `NAME` Anzeigename des Spielers
- `TEAM` Team des Siegers (`ONE`, `TWO`)

```
<room roomId="ROOM_ID">
  <data class="result">
    <definition>
      <fragment name="Gewinner">
        <aggregation>SUM</aggregation>
        <relevantForRanking>true</relevantForRanking>
      </fragment>
      <fragment name="□ Punkte">
        <aggregation>AVERAGE</aggregation>
        <relevantForRanking>true</relevantForRanking>
      </fragment>
    </definition>
    <score cause="CAUSE1" reason="R1">
      <part>WP1</part>
      <part>S1</part>
    </score>
    <score cause="CAUSE2" reason="R2">
      <part>WP2</part>
      <part>S2</part>
    </score>
    <winner displayName="NAME">
      <color class="team">TEAM</color>
    </winner>
  </data>
</room>
```

Spielverlauf

Der Server startet (StandardIp: localhost 13050).

Nun gibt es zwei Varianten ein Spiel zu starten, eine durch einen Administratorclient die andere durch hinzufügen der Spieler zu einen Spieltyp:

Variante 1 (AdminClient Mit Reservierungscode)

Ein Client registriert sich als Administrator mit dem in server.properties festgelegten Passwort pw:

```
<protocol><authenticate password="pw" />
```

Dann kann ein Spiel angelegt werden:

```
<prepare gameType="swc_2021_blokus" pause="true">
  <slot displayName="p1" canTimeout="false" />
  <slot displayName="p2" canTimeout="false" />
</prepare>
```

Der Server antwortet darauf mit einer Nachricht, die die ROOM_ID und Reservierungscodes für die beiden Clients enthält:

```
<protocol>
  <prepared roomId="871faccb-5190-4e44-82fc-6cdcbb493726">
    <reservation>RC1</reservation>
    <reservation>RC2</reservation>
  </prepared>
```

Der Administratorclient kann dann ebenfalls als Observer des Spiels genutzt werden, indem ein entsprechender Request gesendet wird. Dadurch wird das derzeitge Spielfeld (memento) ebenfalls an den Administratorclient gesendet.

```
<observe roomId="871faccb-5190-4e44-82fc-6cdcbb493726" />
```

Clients, die auf dem Serverport (localhost 13050) gestartet werden, können nun mit den Reservierungscodes beitreten.

```
<protocol>
  <joinPrepared reservationCode="RC1" />
```

```
<protocol>
  <joinPrepared reservationCode="RC2" />
```

Variante 2 (kein AdminClient notwendig Ohne Reservierungscode)

Die Clients wurden auf dem Serverport (Standard: localhost 13050) gestartet.

Sie können mit folgender Anfrage einem bereits offenen Spiel des entsprechenden Typs beitreten. Wenn noch keines vorhanden ist, wird dabei automatisch ein neues gestartet.

```
<protocol>
  <join gameType="swc_2019_piranhas" />
```

Der Server antwortet mit:

```
<protocol>
  <joined roomId="871faccb-5190-4e44-82fc-6cdcbb493726" />
```

Weiterer Spielverlauf

Der Server antwortet jeweils mit der WelcomeMessage (Welcome Message) und dem ersten GameState (memento) sobald beide Spieler verbunden sind.

```
<room roomId="cb3bc426-5c70-48b9-9307-943bc328b503">
  <data class="welcomeMessage" color="two"/>
</room>
<room roomId="cb3bc426-5c70-48b9-9307-943bc328b503">
  <data class="memento">
    <state class="state" turn="0" round="1" startPiece="PENTO_V">
      <startTeam class="team">ONE</startTeam>
      <board/>
      <blueShapes class="linked-hash-set">
        <shape>MONO</shape>
        <shape>DOMINO</shape>
        <shape>TRIO_L</shape>
        <shape>TRIO_I</shape>
        <shape>TETRO_O</shape>
        <shape>TETRO_T</shape>
        <shape>TETRO_I</shape>
        <shape>TETRO_L</shape>
        <shape>TETRO_Z</shape>
        <shape>PENTO_L</shape>
        <shape>PENTO_T</shape>
        <shape>PENTO_V</shape>
        <shape>PENTO_S</shape>
        <shape>PENTO_Z</shape>
        <shape>PENTO_I</shape>
        <shape>PENTO_P</shape>
        <shape>PENTO_W</shape>
        <shape>PENTO_U</shape>
        <shape>PENTO_R</shape>
        <shape>PENTO_X</shape>
        <shape>PENTO_Y</shape>
      </blueShapes>
      <yellowShapes class="linked-hash-set">
        <shape>MONO</shape>
        <shape>DOMINO</shape>
        <shape>TRIO_L</shape>
        <shape>TRIO_I</shape>
        <shape>TETRO_O</shape>
        <shape>TETRO_T</shape>
        <shape>TETRO_I</shape>
        <shape>TETRO_L</shape>
        <shape>TETRO_Z</shape>
        <shape>PENTO_L</shape>
        <shape>PENTO_T</shape>
        <shape>PENTO_V</shape>
        <shape>PENTO_S</shape>
```

```
      <shape>PENTO_Z</shape>
      <shape>PENTO_I</shape>
      <shape>PENTO_P</shape>
      <shape>PENTO_W</shape>
      <shape>PENTO_U</shape>
      <shape>PENTO_R</shape>
      <shape>PENTO_X</shape>
      <shape>PENTO_Y</shape>
    </yellowShapes>
    <redShapes class="linked-hash-set">
      <shape>MONO</shape>
      <shape>DOMINO</shape>
      <shape>TRIO_L</shape>
      <shape>TRIO_I</shape>
      <shape>TETRO_O</shape>
      <shape>TETRO_T</shape>
      <shape>TETRO_I</shape>
      <shape>TETRO_L</shape>
      <shape>TETRO_Z</shape>
      <shape>PENTO_L</shape>
      <shape>PENTO_T</shape>
      <shape>PENTO_V</shape>
      <shape>PENTO_S</shape>
      <shape>PENTO_Z</shape>
      <shape>PENTO_I</shape>
      <shape>PENTO_P</shape>
      <shape>PENTO_W</shape>
      <shape>PENTO_U</shape>
      <shape>PENTO_R</shape>
      <shape>PENTO_X</shape>
      <shape>PENTO_Y</shape>
    </redShapes>
    <greenShapes class="linked-hash-set">
      <shape>MONO</shape>
      <shape>DOMINO</shape>
      <shape>TRIO_L</shape>
      <shape>TRIO_I</shape>
      <shape>TETRO_O</shape>
      <shape>TETRO_T</shape>
      <shape>TETRO_I</shape>
      <shape>TETRO_L</shape>
      <shape>TETRO_Z</shape>
      <shape>PENTO_L</shape>
      <shape>PENTO_T</shape>
      <shape>PENTO_V</shape>
      <shape>PENTO_S</shape>
      <shape>PENTO_Z</shape>
      <shape>PENTO_I</shape>
      <shape>PENTO_P</shape>
      <shape>PENTO_W</shape>
      <shape>PENTO_U</shape>
```

```
        <shape>PENTO_R</shape>
        <shape>PENTO_X</shape>
        <shape>PENTO_Y</shape>
      </greenShapes>
      <lastMoveMono class="linked-hash-map"/>
      <validColors class="linked-hash-set">
        <color>BLUE</color>
        <color>YELLOW</color>
        <color>RED</color>
        <color>GREEN</color>
      </validColors>
      <first displayName="One">
        <color class="team">ONE</color>
      </first>
      <second displayName="Two">
        <color class="team">TWO</color>
      </second>
    </state>
  </data>
</room>
```

Der erste Spieler erhält dann eine Zugaufforderung ([move-request]), falls in server.properties paused auf false gesetzt wurde. Falls das Spiel pausiert ist, muss das Spiel durch einen Administratorclient gestartet werden:

Verbinden des Administratorclients (falls es sich um die erste Kontaktaufnahme zum Server handelt, ansonsten <protocol> weglassen).

```
<protocol>
  <authenticate password="examplepassword" />
```

Pausierung aufheben:

```
<pause roomId="871faccb-5190-4e44-82fc-6cdcbb493726" pause="false" />
```

Daraufhin wird der erste Spieler aufgefordert einen Zug zu senden:

```
<room roomId="871faccb-5190-4e44-82fc-6cdcbb493726">
  <data class="sc.framework.plugins.protocol.MoveRequest" />
</room>
```

Der Client des CurrentPlayer sendet nun einen Zug (ZUG):

```
<room roomId="cb3bc426-5c70-48b9-9307-943bc328b503">
  <data class="sc.plugin2021.SetMove">
    <piece color="BLUE" kind="PENTO_V" rotation="RIGHT" isFlipped="false">
      <position x="17" y="0"/>
    </piece>
  </data>
</room>
```

So geht es abwechselnd weiter, bis zum Spielende (Spielergebnis). Die letzte Nachricht des Servers endet mit:

```
</protocol>
```

Danach wird die Verbindung geschlossen.