Ostseeschach XML-Dokumentation der Software-Challenge 2022

Inhalt

Dieses Dokument erklärt die XML-Schnittstelle der Software-Challenge für das aktuelle Spiel.

Beitragen

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. Damit wird von GitHub automatisch ein Fork und ein Pull Request erstellt.
Alternativ auch gern eine E-Mail an info@software-challenge.de oder eine Nachricht im Discord.

Spiel betreten

```
<protocol>
```

Dieses Tag eröffnet die Kommunikation mit dem Server. Dann gibt es drei Möglichkeiten, einer Spielpartie beizutreten.

Beliebige Partie

Betritt ein beliebige offene Spielpartie:

```
<join />
```

Sollte kein Spiel offen sein, wird automatisch ein neues erstellt. Abhängig von der Einstellung `paused` in `server.properties` wird das Spiel pausiert gestartet oder nicht.

Bestimmte Partie

Einer bestimmten, bereits offenen aber noch nicht gestarteten Partie kann durch Angabe der `ROOM_ID` beigetreten werden:

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

Mit Reservierung

Unter Angabe eines Reservierungscodes kann man einen reservierten Platz in einer geplanten Partie einnehmen:

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

Antwort

Der Server antwortet auf einen erfolgreichen Spielbeitritt mit:

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

ROOM_ID
: Identifikationscode der Spielpartie

Spielverlauf

Start

Der Server eröffnet das Spiel mit einer Begrüßung und dem initialen Spielstatus sobald beide Spieler verbunden sind.

ROOM_ID
: Identifikationscode der Spielpartie

COLOR

Spielerfarbe

STATUS

Spielstatus der Partie

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

Ablauf

Der erste Spieler erhält dann eine Zugaufforderung:

```
<room roomId="ROOM_ID">
  <data class="moveRequest" />
</room>
```

Worauf dieser innerhalb der gesetzten Zeitbeschränkung mit einem Zug antwortet:

```
<room roomId="ROOM_ID">
  <data class="move">
    ZUG
  </data>
</room>
```

Nach Erhalt des Zuges sendet der Server den neuen Spielstatus an alle Spieler und dem nächsten Spieler eine Zugaufforderung. So geht es hin und her bis zum Spielergebnis.

Ende der Kommunikation

Die letzte Nachricht des Servers endet mit:

```
</protocol>
```

Danach wird die Verbindung geschlossen.

Spielstatus

startTeam

das Team, das als erstes dran ist

board

die Positionen der Spielfiguren - `count` gibt die Turmhöhe an

lastMove

der vorherige Zug - hat die selbe Struktur wie ein Zug, der gesendet wird, außer dass das Tag `<lastMove>` statt `<move>` heisst. Der vorherige Zug wird in jedem Spielstatus angegeben, ausser vor dem ersten Zug.

ambers

die Anzahl der Bernsteine pro Team

```
<state turn="27">
  <startTeam>ONE</startTeam>
  <board>
    <pieces>
      <entry>
        <coordinates x="0" y="0"/>
        <piece type="Moewe" team="TWO" count="1"/>
      </entry>
      <entry>
        <coordinates x="5" y="6"/>
        <piece type="Robbe" team="ONE" count="2"/>
      </entry>
    </pieces>
  </board>
  <lastMove>
    <from x="4" y="4"/>
    <to x="5" y="6"/>
  </lastMove>
  <ambers>
    <entry>
      <team>ONE</team>
      <int>1</int>
    </entry>
    <entry>
      <team>TWO</team>
      <int>0</int>
    </entry>
  </ambers>
</state>
```

Zug

`from`

Koordinaten des zu ziehenden Spielsteines

`to`

Koordinaten des Zielfeldes

```
<from x="0" y="1"/>
<to x="1" y="2"/>
```

Spielergebnis

Zum Spielende erhält jeder Spieler das Ergebnis. Es beginnt mit einer `definition`, die die Interpretation der Ergebnisse erklärt. Für jeden Spieler gibt es einen Eintrag in `scores`. Der darin enthaltene `score` schlüsselt sich auf in:

cause

Beitrag des Spielers zum Spielende (`REGULAR`, `LEFT`, `RULE_VIOLATION`, `SOFT_TIMEOUT`, `HARD_TIMEOUT`)

reason

Erklärung zu `cause`

part

Siegpunkte des Spielers (0 verloren, 1 unentschieden, 2 gewonnen) und weitere Punkteinträge entsprechend `definition`

Wenn es einen Sieger gibt, endet es mit einem `winner`-Tag, welches das Gewinner-Team angibt.

Hier ein Beispiel:

```
<room roomId="ROOM_ID">
  <data class="result">
    <definition>
      <fragment name="Siegpunkte">
        <aggregation>SUM</aggregation>
        <relevantForRanking>true</relevantForRanking>
      </fragment>
      <fragment name="□ Punkte">
        <aggregation>AVERAGE</aggregation>
        <relevantForRanking>true</relevantForRanking>
      </fragment>
    </definition>
    <scores>
      <entry>
        <player name="rad" team="ONE"/>
        <score cause="REGULAR" reason="">
          <part>2</part>
          <part>27</part>
        </score>
      </entry>
      <entry>
        <player name="blues" team="TWO"/>
        <score cause="LEFT" reason="Player left">
          <part>0</part>
          <part>15</part>
        </score>
      </entry>
    </scores>
    <winner team="ONE"/>
  </data>
</room>
```