Software Challenge XML-Dokumentation Hase und Igel

Sören Domrös

Inhalt

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.

1. 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.

1.1. 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.

1.2. 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 mit einem newline abgeschlossen.

2. Spiel betreten

Wenn begonnen wird mit dem Server zu kommunizieren, muss zuallererst

```
<protocol>
```

gesendet werden, um die Kommunikation zu beginnen.

2.1. Ohne Reservierung

Betritt ein beliebiges offenes Spiel:

```
<join gameType="swc_2018_hase_und_igel"/>
```

Sollte kein Spiel offen sein, wird so ein neues erstellt. Der Server antwortet darauf mit:

- ROOM_ID Id des GameRooms

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

2.2. Mit Reservierungscode

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

2.2.1. Join mit RC

- RC Reservierungscode

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

2.2.2. Welcome Message

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

- ROOM_ID Id des GameRooms
- COLOR Spielerfarbe also red oder blue
- status GameState wie in Status

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

3. Züge senden

3.1. Der Move

Der Move ist die Antwort auf den MoveRequest des Servers.

3.1.1. MoveRequest

- ROOM_ID Id des GameRooms

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

3.1.2. Senden

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

- ROOM_ID Id des GameRooms
- ZUG Zug wie in ZUG

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

3.2. ZUG

- ACTION Aktionen wie in ACTION

```
<data class="move">
  ACTION
  ..
  ACTION
</move>
```

3.3. ACTION

Mögliche Aktionen:

- I Index der Aktion beginnend mit 0
- D Anzahl der Felder um die sich bewegt wird
- W -10 oder 10
- V 20,0,-20 falls type TAKE_OR_DROP_CARROTS, 0 sonst

```
<advance order="I" distance="D"/>
<card order="I" type="CARD_TYPE" value="V"/>
<exchangeCarrots order="I" value="W"/>
<eatSalad order="I"/>
<fallBack order="I"/>
<skip order="I"/>
```

3.4. Debughints

Zügen können Debug-Informationen beigefügt werden:

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

Damit sieht beispielsweise ein Laufzug so aus:

```
<room roomId="ROOM_ID">
  <data class="move">
    <advance order="0" distance="1"/>
    <card order="1" type="EAT_SALAD" value="0"/>
    <hint content="Dies ist ein Hint."/>
    <hint content="noch ein Hint"/>
  </data>
</room>
```

4. 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:

4.1. memento

- ROOM_ID Id des GameRooms
- status Gamestate wie in Status

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

4.2. Status

- Z aktuelle Zugzahl
- S Spieler, der das Spiel gestartet hat (RED/BLUE)
- C Spieler, der an der Reihe ist (RED/BLUE)
- red, blue wie in Spieler definiert
- board Das Spielbrett, wie in Spielbrett definiert
- lastMove Letzter getätigter Zug (nicht in der ersten Runde), wie in Letzter Zug definiert
- condition Spielergebnis, wie in Spielergebnis definiert; nur zum Spielende

```
<state class="state" turn="Z" startPlayer="S" currentPlayer="C">
  red
  blue
  [board]
  [lastMove]
  [condition]
</state>
```

4.3. Spieler

- C Farbe (red/blue)
- N Anzeigename
- I Feldindex des Spielers
- S Anzahl Salate
- K Anzahl der Karotten

```
<C displayName="N" color="C" index="I" carrots="K" salads="S">
  <cards>
    <type>TAKE_OR_DROP_CARROTS</type>
    <type>EAT_SALAD</type>
    <type>HURRY_AHEAD</type>
    <type>FALL_BACK</type>
  </cards>
  <lastNonSkipAction class="fallBack" order="0"/> // Beispiel für letzte Aktion
</C>
```

4.4. Spielbrett

- FIELD Ein Spielfeld wie in Spielfeld definiert.

```
<board>
  <fields index="0" type="START"/>
  ..
  FIELD
  ..
  <fields index="64" type="GOAL"/>
</board>
```

4.5. Spielfeld

- I Feldnummer
- TYPE Typ des Feldes (START/CARROT/HARE/GOAL/POSITION_{X}/HEDGEHOG/SALAD). X ist 1 oder 2

```
<fields index="I" type="TYPE"/>
```

4.6. Letzter Zug

Der letzte Zug ist ein Move (siehe hierzu ZUG).

- ACTIONS Teilzüge des Zuges (siehe hierzu ACTION).

```
<lastMove>
  ACTIONS
</lastMove>
```

5. Zug-Anforderung

Eine einfache Nachricht fordert zum Zug auf:

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

6. Fehler

Ein “spielfähiger” Client muss nicht mit Fehlern umgehen können. Fehlerhafte Züge beispielsweise resultieren in einem vorzeitigen Ende des Spieles, das im nächsten gesendeten Gamestate erkannt werden kann (siehe Spielergebnis).

- MSG Fehlermeldung

```
<room roomId="RID">
  <error message="MSG">
    <originalRequest>
    Request, der den Fehler verursacht hat
    </originalRequest>
  </error>
</room>
```

7. 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"/>
```

8. 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
- I1, I2 Index des Feldes auf dem der Spieler steht
- C1, C2 Karotten des Spielers
- NAME Anzeigename des Spielers
- COLOR Farbe des Siegers
- I3 I1 oder I2 je nachdem wer gewonnen hat
- C3 C1 oder C2 je nachdem wer gewonnen hat
- S Salate des Siegers
- [cards] Karten des Siegers wie in Spieler
- [lastNonSkipAction] Letztes Aktion des Siegers wie in Spieler

```
<room roomId="ROOM_ID">
  <data class="result">
    <definition>
      <fragment name="Gewinner">
        <aggregation>SUM</aggregation>
        <relevantForRanking>true</relevantForRanking>
      </fragment>
      <fragment name="Ø Feldnummer">
        <aggregation>AVERAGE</aggregation>
        <relevantForRanking>true</relevantForRanking>
      </fragment>
      <fragment name="Ø Karotten">
        <aggregation>AVERAGE</aggregation>
        <relevantForRanking>true</relevantForRanking>
      </fragment>
    </definition>
    <score cause="CAUSE1" reason="R1">
      <part>WP1</part>
      <part>I1</part>
      <part>C1</part>
    </score>
    <score cause="CAUSE2" reason="R2">
      <part>WP2</part>
      <part>I2</part>
      <part>C1</part>
    </score>
    <winner class="player" displayName="NAME" color="COLOR" index="I3" carrots="C3"
salads="S">
      [cards]
      [lastNonSkipAction]
    </winner>
  </data>
</room>
```

9. 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:

9.1. Variante 1 (AdminClient Mit Reservierungscode)

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

```
<protocol><authenticate passphrase="p"/>
```

Dann kann ein Spiel angelegt werden:

```
<prepare gameType="swc_2018_hase_und_igel">
  <slot displayName="p1" canTimeout="false" shouldBePaused="true"/>
  <slot displayName="p2" canTimeout="false" shouldBePaused="true"/>
</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 nur 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 so über diesen Code joinen.

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

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

9.2. Variante 1 (AdminClient Mit Reservierungscode)

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

Sie können sich mit folgender Anfrage einen bereits offenen Spiel gleichen Typs beitreten oder, falls kein Spiel des Typs vorhanden selbst eines starten:

```
<protocol>
  <join gameType="swc_2018_hase_und_igel"/>
```

Der Server antwortet mit:

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

9.3. Weiterer Spielverlauf

Der Server antwortet jeweils mit der WelcomeMessage (]) und dem ersten GameState (xref:memento[) sobald beide Spieler verbunden sind.

```
<room roomId="871faccb-5190-4e44-82fc-6cdcbb493726">
  <data class="welcomeMessage" color="red"/>
</room>
<room roomId="871faccb-5190-4e44-82fc-6cdcbb493726">
  <data class="memento">
    <state class="state" turn="0" startPlayer="RED" currentPlayer="RED">
      <red displayName="Unknown" color="RED" index="0" carrots="68" salads="5">
        <cards>
          <type>TAKE_OR_DROP_CARROTS</type>
          <type>EAT_SALAD</type>
          <type>HURRY_AHEAD</type>
          <type>FALL_BACK</type>
        </cards>
      </red>
      <blue displayName="Unknown" color="BLUE" index="0" carrots="68" salads="5">
        <cards>
          <type>TAKE_OR_DROP_CARROTS</type>
          <type>EAT_SALAD</type>
          <type>HURRY_AHEAD</type>
          <type>FALL_BACK</type>
        </cards>
      </blue>
      <board>
        <fields index="0" type="START"/>
        <fields index="1" type="CARROT"/>
        <fields index="2" type="HARE"/>
        <fields index="3" type="HARE"/>
        <fields index="4" type="POSITION_2"/>
        <fields index="5" type="POSITION_1"/>
        <fields index="6" type="CARROT"/>
        <fields index="7" type="CARROT"/>
        <fields index="8" type="HARE"/>
        <fields index="9" type="CARROT"/>
        <fields index="10" type="SALAD"/>
        <fields index="11" type="HEDGEHOG"/>
        <fields index="12" type="HARE"/>
        <fields index="13" type="CARROT"/>
        <fields index="14" type="CARROT"/>
        <fields index="15" type="HEDGEHOG"/>
        <fields index="16" type="POSITION_1"/>
```

```
        <fields index="17" type="CARROT"/>
        <fields index="18" type="POSITION_2"/>
        <fields index="19" type="HEDGEHOG"/>
        <fields index="20" type="CARROT"/>
        ...
        <fields index="57" type="SALAD"/>
        <fields index="58" type="CARROT"/>
        <fields index="59" type="POSITION_1"/>
        <fields index="60" type="HARE"/>
        <fields index="61" type="CARROT"/>
        <fields index="62" type="HARE"/>
        <fields index="63" type="CARROT"/>
        <fields index="64" type="GOAL"/>
      </board>
    </state>
  </data>
</room>
```

Der erste Spieler erhält dann eine Zugaufforderung (MoveRequest):

```
<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="871faccb-5190-4e44-82fc-6cdcbb493726">
  <data class="move">
    <advance order="0" distance="6">
  </data>
</room>
```

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

```
</protocol>
```

Danach wird die Verbindung geschlossen.