# Dokumentation Piranhas Software Challenge 2019

Sven Koschnicke

# Spielregeln Piranhas: Software Challenge 2019

Sven Koschnicke <svk@informatik.uni-kiel.de>

**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. Alternativ auch gern eine E-Mail an svk@informatik.uni-kiel.de.

Das Spielfeld ist ein Aquarium (10x10 Schachbrett mit Spalten A-J und Zeilen 0-9), ein Spieler spielt die roten, der andere die blauen Fische.

## Anfangsaufstellung

- 8 rote Fische auf den (Schachbrett-)Feldern A1-A8
- 8 rote Fische auf den Feldern J1-J8
- 8 blaue Fische auf den Feldern B0-I0
- 8 blaue Fische auf den Feldern B9-I9

Auf dem Spielfeld gibt es im inneren 6x6-Bereich zwei zufällig generierte Kraken-Felder, die zwar durchschwommen werden können, auf denen jedoch kein Zug enden darf. Diese Felder behindern also die Schwarmbildung. Die beiden zufällig generierten Felder befinden sich weder in derselben Zeile noch in derselben Spalte, noch in derselben Diagonale.

## Ziel des Spieles

Alle Fische der eigenen Farbe sollen zu einem Schwarm verbunden werden, sie sollen also auf einem zusammenhängenden Gebiet auf dem Spielfeld stehen, wobei senkrecht, waagerecht und diagonal nebeneinanderliegende Felder als zusammenhängend gelten.

## Züge

Es wird abwechselnd gezogen, wobei Rot beginnt. Jede Runde macht jeder Spieler einen Zug. In jedem Zug muss man einen eigenen Fisch auf einer senkrechten, waagerechten oder diagonalen Linie genau so viele Felder in eine Richtung bewegen, wie insgesamt Fische (beider Farben) auf dieser Linie sind.

Dabei muss man auf einem leeren Feld oder auf einem vom Gegner besetzten Feld landen. Beim Zug darf man Felder mit eigenen Fischen sowie Kraken überspringen, nicht aber Felder mit gegnerischen Fischen. Bei der Anzahl der zu ziehenden Felder werden übersprungene Felder

mitgezählt. Wenn man auf einem Feld ankommt, auf dem ein gegnerischer Fisch schwimmt, wird dieser gefressen (ersatzlos vom Spielfeld entfernt).

Zum Beispiel kann Rot im 1. Zug einen beliebigen roten Fisch zwei Felder in Richtung der Spielfeldmitte bewegen, z.B. von A4 nach C4. Eine andere Möglichkeit für den 1. Zug wäre, einen Fisch von A1 nach A9 zu ziehen. Man könnte auch mit dem Fisch auf A2 den gegnerischen Fisch auf C0 fressen.

## Spielende

Das Spiel endet, wenn am Ende einer Runde alle auf dem Spielfeld verbliebenen Fische einer Farbe zu einem einzigen zusammenhängenden Schwarm verbunden sind, spätestens jedoch nach 30 Runden. (Blau könnte also einen roten Schwarm noch in der 2. Hälfte einer Runde wieder zerstören.) Das Spiel endet ebenfalls, wenn einer der Spieler keinen regulären Zug machen kann. In diesem Fall hat dieser Spieler verloren.

## Gewinner

Falls beim Spielende die Fische nur eines Spielers zu einem einzigen zusammenhängenden Schwarm verbunden sind, hat dieser Spieler gewonnen; anderenfalls derjenige Spieler mit dem größeren Schwarm. Falls nach den vorgenannten Kriterien kein Gewinner ermittelt werden kann, endet das Spiel unentschieden.

Nachrangiges Kriterium für das Ranking im Wettkampfsystem: Anzahl der Fische im größten Schwarm des Spielers.

Beim Final Eight gibt es kein nachrangiges Kriterium. Beim Final Eight gewinnt derjenige Spieler eine Begegnung, der nach einer geraden Anzahl von Spielen (mindestens 6) mehr Spiele als der Gegner gewonnen hat.

# Software Challenge XML-Dokumentation Piranhas

Sören Domrös <stu114053@mail.uni-kiel.de>

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 mit einem newline abgeschlossen.

# 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_2019_piranhas"/>
```

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 RC

- RC Reservierungscode

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

#### 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>
```

# Züge senden

## Der Move

Der Move ist die Antwort auf den MoveRequest des Servers.

#### MoveRequest

`ROOM_ID`
: ID des GameRooms, dient der Zuordnung der Antworten des Clients.

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

#### 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

`X`
: X-Koordinate des zu ziehenden Piranhas (`0` bis `9`, Spalten des Spielfeldes von links nach rechts)

`Y`
: Y-Koordinate des zu ziehenden Piranhas (`0` bis `9`, Zeilen des Spielfeldes von unten nach oben)

`DIRECTION`
: Richtung, in die der Piranha zieht. Eine von folgenden:

  - `UP`
  - `UP_RIGHT`
  - `RIGHT`
  - `DOWN_RIGHT`
  - `DOWN`
  - `DOWN_LEFT`
  - `LEFT`
  - `UP_LEFT`

```
<data class="move" x="X" y="Y" direction="DIRECTION"/>
```

## Debughints

Zügen können Debug-Informationen beigefügt werden:

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

Damit sieht beispielsweise ein Laufzug so aus:

```
<room roomId="ROOM_ID">
  <data class="move" x="0" y="0" direction="UP">
    <hint content="Dies ist ein Hint."/>
    <hint content="noch ein Hint"/>
  </data>
</room>
```

# 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
- 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>
```

### Spieler

- C Farbe (red/blue)

```
<C displayName="N" color="C"/>
```

### Spielbrett

- FIELD Ein Spielfeld wie in Spielfeld definiert.

```
<board>
  <fields x="0" y="0" state="EMPTY"/>
  ..
  FIELD
  ..
  <fields x="9" y="9" state="EMPTY"/>
</board>
```

### Spielfeld

- X X-Koordinate
- Y Y-Koordinate
- STATE Zustand des Feldes (RED,BLUE,OBSTRUCTED,EMPTY)

```
<fields x="X" y="Y" type="STATE"/>
```

### Letzter Zug

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

```
<lastMove>
  ZUG
</lastMove>
```

# Zug-Anforderung

Eine einfache Nachricht fordert zum Zug auf:

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

# 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>
```

# 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
- NAME Anzeigename des Spielers

- COLOR Farbe des Siegers
- S3 S1 oder S2 je nachdem wer gewonnen hat

```
<room roomId="ROOM_ID">
  <data class="result">
    <definition>
      <fragment name="Gewinner">
        <aggregation>SUM</aggregation>
        <relevantForRanking>true</relevantForRanking>
      </fragment>
      <fragment name="Ø Schwarm">
        <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 class="player" displayName="NAME" color="COLOR"/>
  </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 p:

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

Dann kann ein Spiel angelegt werden:

```
<prepare gameType="swc_2019_piranhas">
  <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"/>
```

### Variante 2 ((eventuell) ohne AdminClient Ohne 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_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="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"/>
      <blue displayName="Unknown" color="BLUE"/>
      <board>
        <fields x="0" y="0" state="EMPTY"/>
    TODO
        <fields x="9" y="9" state="EMPTY"/>
      </board>
    </state>
  </data>
</room>
```

Der erste Spieler erhält dann eine Zugaufforderung (MoveRequest), 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 passphrase="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="871faccb-5190-4e44-82fc-6cdcbb493726">
  <data class="move" x="0" y="0" direection="UP"/>
</room>
```

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

```
</protocol>
```

Danach wird die Verbindung geschlossen.