# Phoenixtool Server API Dokumentation

Roberts Kolosovs

21. November 2017

## Einleitung

Dieses Dokument beschreibt die Schnittstelle des Phönixtool-Servers. Der Informationsaustausch mit dem Server erfolgt im JSON-Format [1].
Abschnitt 1 beschreibt die Repräsentation des Spielstandes und dessen Auslesen. Im Abschnitt 2 erklären wir, wie man einen neuen Spielstand auf den Server schreibt. Der Abschnitt 3 beschreibt unser Sicherheitskonzept und den Login-View. Die Struktur unserer Spielzüge ist beschrieben im Abschnitt 4. Die darin verwendeten Events werden einzeln im Abschnitt 5 beschreiben.

## 1 Lesen von Spieldaten

In diesem Abschnitt beschreiben wir unsere Repräsentation des Spielstandes und den Format, in dem es vom Server geliefert wird. Der Unterabschnitt 1.1 beschreibt das Format, in dem Armeen und Flotten geliefert werden. Die Gebäude sind im Unterabschnitt 1.2. Die Geländetypen des Spielfelds sind in 1.3 und die Flüsse sind in 1.4 beschrieben. Zu Letzt sind die Gebiete der Reiche in 1.5 beschreiben.

### 1.1 Armeen

Die Heere (Armeen) und Flotten erhält man mit einer Anfrage an "/databaseLink/armydata/". Die Anfrage muss wie folgt aussehen:

```
{'authorization': Auth-Token}
```

Das Authentifizierungs-Token identifiziert einen eingeloggten Nutzer und ist in 3 beschrieben. Die Antwort hat das folgende Format:

```
{
  'realm': Kontrolierendes Reich (DB-Key),
  'armyId': Armeenummer (Phoenix),
  'count': Truppen-/Schiffsanzahl,
  'leaders': Heerführeranzahl,
  'mounts': mitgeführte Reittiere,
  'lkp': leichte Katapulte/Kriegsschiffe,
  'skp': schwere Katapulte/Kriegsschiffe,
  'x': x-Koordinate der Position,
  'y': y-Koordinate der Position,
  'isGuard': True für Gardeheere/ -flotten,
  'isLoadedIn': True für Landheere auf Schiffen,
  'movementPoints': Bewegungspunkte der Armee,
  'heightPoints': Höhenpunkte der Armee
}
```

Fußheere, Reiterheere und Flotten werden Serverseitig gleich gespeichert. Welches gemeint ist und somit, welche Bedeutung die einzelnen Felder haben, lässt sich aus dem 'armyId'-Feld ablesen. Die Fußheere haben Nummern 101-199, Reiterheere 201-299 und Flotten 301-399. Je nach dem, als welchen Nutzer der Authentifizierungs-Token einen identifiziert, bekommt man verschiedene Informationen. Ein Spielleitungs-Nutzer bekommt die richtigen Anzahlen von allen Armeen/Flotten. Ein eingeloggter Spieler bekommt richtige Zahlen für alle Armeen seines Reiches. Für fremde Armeen wird 'isGuard' auf False gesetzt. Die Felder 'count', 'leaders', 'mounts', 'lkp' und 'skp' werden auf -1 gesetzt, was sonst ein ungültiger Wert ist. Für sonstige Nutzer, auch nicht eingeloggte, werden alle Armeen auf diese Art

[1] http://www.json.org/

maskiert.

## 1.2 Gebäude

Rüstorte und andere Gebäude werden mit einer Anfrage an "/databaseLink/buildingdata/". Die Antwort kommt in folgender Form:

```
[{
  'realm': Kontrolierendes Reich (DB-Key),
  'name': benutzerdefinierter Name,
  'type': 0 - 8,
  'x': x-Koordinate (außer Straße),
  'y': y-Koordinate (außer Straße),
  'direction': "nw"/"ne"/"e"/"se"/"sw"/"w",
  'firstX': x-Koordinate Straßenanfang,
  'firstY': y-Koordinate Straßenanfang,
  'secondX': x-Koordinate Straßenende,
  'secondY': y-Koordinate Straßenende
}, ...]
```

Das 'realm'-Feld ist redundant, da der Besitzer eines Rüstortes mit dem Besitzer des Feldes identisch ist. In der Zukunft sollte es entweder entfernt oder durch den Reichskürzel ersetzt werden. Der benutzerdefinierte Name ist rein kosmetisch und wird nur benutzt, falls einzelne Rüstorte besondere Namen haben sollen. Dies sollte *nicht* benutzt werden, um einzelnen Rüstorten besondere Funktion zu verleihen. Dazu sollten eigene Felder oder Datenbanktabellen angelegt werden. Die Typen sind von 0 bis 8 wie folgt durchnummeriert:

- 0. Burg
- 1. Stadt
- 2. Festung
- 3. Hauptstadt
- 4. Festungshauptstadt
- 5. Mauer
- 6. Kaianlage
- 7. Brücke
- 8. Straße

Alle Gebäude außer Straßen haben "null" als Werte für die Felder 'firstX', 'firstY', 'secondX' und 'secondY'. Ihre Position wird durch die Felder 'x' und 'y' angegeben. Straßen haben "null" als Wert der Felder 'x' und 'y' und werden zwischen den Feldern ('firstX', 'firstY') und ('secondX', 'secondY') gezogen.
Mauern, Anlegestellen und Brücken haben im Feld 'direction' eine Richtung, in die sie von ihrem Feld aus zeigen. Die Richtung ist angegeben als Ein- oder Zwei-Buchstaben Kürzel der Himmelsrichtung, wo die Spitzen der Hex-Felder nach Norden und Süden zeigen. Die x-Werte des Koordinatensystems erhöhen sich von Norden nach Süden. Die y-Werte erhöhen sich von Westen nach Osten. Für andere Gebäudetypen ist dieser Wert "null", da es entweder irrelevant ist (Burg, Stadt, Festung, (Festungs)hauptstadt) oder durch ihre mit zwei Koordinatenpaaren angegebene Position gegeben ist (Straße).

## 1.3 Spielfeld

Die Geländetypen erhält man mit einer Anfrage an "/databaseLink/fielddata/". Die Antwort sieht wie folgt aus:

```
[{
  'type': 0 - 8,
  'x': x-Koordinate,
  'y': y-Koordinate
}, ...]
```

Die Typen sind von 0 bis 8 wie folgt durchnummeriert:

- 0. See
- 1. Tiefsee
- 2. Tiefland
- 3. Tieflandwald
- 4. Hochland
- 5. Bergland

- 6. Gebirge
- 7. Tieflandwüste
- 8. Tieflandsumpf

Andere Werte können benutzt werden, um undefinierte Felder zu beschreiben: Das Feld existiert, hat aber noch keinen Geländetyp zugewiesen.
Das Koordinatensystem hat ihren Ursprung in der Mitte der Karte. Das obere linke Quadrant hat negative x- und y-Werte. Das obere rechte Quadrant hat positive x- und negative y-Werte. Das untere rechte Quadrant hat positive x- und y-Werte und das untere linke Quadrant hat negative x- und positive y-Werte.
Die Zeilen mit ungeraden y-Werten sind den Zeilen mit geraden y-Werten gegenüber um ein halbes Feld nach links verschoben. Da es sich um Hex-Felder handelt, ergeben sich die Nachbarschaftsverhältnisse in den Abbildungen 1 und 2.

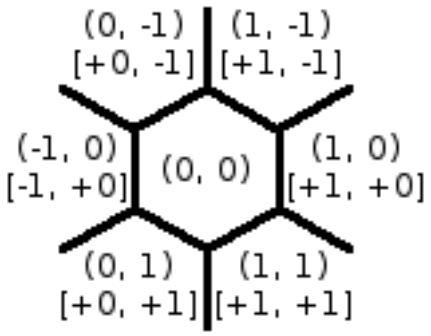


Abbildung 1: Nachbarschaft geradzahlige Zeilen

## 1.4 Flüsse

Um die Flüsse zu erhalten, muss man eine Anfrage an "/databaseLink/getriverdata/" schicken. Die Antwort kommt in folgender Form:

```
[{
  'firstX': x-Koordinate 1. Feld,
  'firstY': y-Koordinate 1. Feld,
  'secondX': x-Koordinate 2. Feld,
  'secondY': y-Koordinate 2. Feld
}, ...]
```

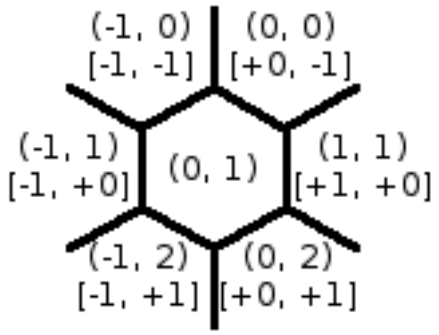


Abbildung 2: Nachbarschaft ungeradzahlige Zeilen

Da Flüsse immer auf den Kanten zwischen zwei Feldern sind, werden sie durch die x- und y-Koordinaten dieser zwei Felder beschreiben.

## 1.5 Grenzen

Die Reichsgebiete erhält man durch eine Anfrage an "/databaseLink/getborderdata/". Das Antwortformat ist wie folgt:

```
[{
  'tag': Kontrolierendes Reich (Kürzel),
  'land': [
    [x-Koordinate, y-Koordinate], ...
  ]
}, ...]
```

Die Reichsgrenzen sind als Arrays von Feldern angegeben, wobei jedes Feld ein Array von zwei Koordinaten ist. Die Felder sind nach Reichen sortiert. Das heißt, das Array der Reichsgrenzen enthält ein Objekt für jedes aktive Reich. Diese Objekte enthalten die Felder 'tag', welches den drei-Buchstaben-Kürzel des Reiches enthält, und 'land', welches das oben beschriebene Array von Feldern enthält.

# 2 Schreiben von Spieldaten

Dieser Abschnitt beschreibt, mit welchen Anfragen und in welchem Format die Änderungen an dem Spielstand auf den Server geschrieben werden können. Analog zum Abschnitt 1 beschreiben die Unterabschnitte 2.1, 2.2, 2.3, 2.4 und 2.5 die Formate für Armeen/Flotten, Gebäude, Geländetypen, Flüsse und Reichsgebiete.

## 2.1 Armeen

Man speichert die Armeen in dem man eine Anfrage an "/databaseLink/savearmydata/" schickt. Die Anfrage muss wie folgt aussehen:

```
{
  'armies': [
    'armyId': Armeenummer (Phoenix),
  'count': Truppen-/Schiffsanzahl,
    'leaders': Heerführerzahl,
    'lkp': leichte Katapulte/Kriegsschiffe,
    'skp': schwere Katapulte/Kriegsschiffe,
    'mounts': mitgeführte Reittiere,
    'ownerPk': Kontrolierendes Reich
               (DB-Key),
    'movementPoints': Bewegungspunkte
                      der Armee,
    'heightPoints': Höhenpunkte
                    der Armee,
    'isLoadedIn': True für Landheere
                  auf Schiffen
  ],
  'authorization': Auth-Token
}
```

Das Feld 'armies' enthält die zu schreibenden Armeen als ein Array, das mit dem Stringify-Verfahren zu einem String verpackt wurde. Soll eine Armee gelöscht werden, lässt man sie einfach aus der Liste raus. In diesem Fall wird die Armee serverseitig nicht aus der DB entfernt, aber als 'inactive' markiert. So markierte Armeen sind effektiv gelöscht: Sie werden nicht mit den restlichen Armeen an den Client geschickt und werden nur zur Spielhistorie in der DB behalten. Das 'authorization'-Feld enthält den Authentifizierungs-Token. Nur wenn dieser den User als einen Spielleiter identifiziert, wird diese Anfrage akzeptiert.
Die Anfrage liefert Statuscodes mit folgender Bedeutung zurück:

- 200: Erfolg
- 401: Nutzer nicht eingeloggt
- 403: Eingeloggter Nutzer hat keine SL-Rechte

## 2.2 Gebäude

Die Gebäude werden mit der Anfrage an "/databaseLink/savebuildingdata/" gespeichert. Der Inhalt der Anfrage muss wie folgt aussehen:

```
{
  'buildings': Gebäude als String,
  'authorization': Auth-Token
}
```

Das Feld 'buildings' enthält die zu schreibenden Gebäude als ein String. **Nicht maintainable! Dies sollte schnellstmöglich zu einem vernünftigen JSON-Format geändert werden!** Das 'authorization'-Feld enthält den Authentifizierungs-Token. Nur wenn dieser den User als einen Spielleiter identifiziert, wird diese Anfrage akzeptiert.
Die Anfrage liefert Statuscodes mit folgender Bedeutung zurück:

- 200: Erfolg
- 401: Nutzer nicht eingeloggt
- 403: Eingeloggter Nutzer hat keine SL-Rechte

## 2.3 Spielfeld

Die Gebäude werden mit der Anfrage an "/databaseLink/savefielddata/" gespeichert. Der Inhalt der Anfrage muss wie folgt aussehen:

```
{
  'map': Kartenfelder als String,
  'authorization': Auth-Token
}
```

Das Feld 'map' enthält die zu schreibenden Felder als ein String. **Nicht maintainable! Dies sollte schnellstmöglich zu einem vernünftigen JSON-Format geändert werden!** Das 'authorization'-Feld enthält den Authentifizierungs-Token. Nur wenn dieser den User als einen Spielleiter identifiziert, wird diese Anfrage akzeptiert.
Die Anfrage liefert Statuscodes mit folgender Bedeutung zurück:

- 200: Erfolg
- 401: Nutzer nicht eingeloggt
- 403: Eingeloggter Nutzer hat keine SL-Rechte

### 2.4 Flüsse

Die Gebäude werden mit der Anfrage an "/databaseLink/saveriverdata/" gespeichert. Der Inhalt der Anfrage muss wie folgt aussehen:

```
{
  'river': Flüsse als String,
  'authorization': Auth-Token
}
```

Das Feld 'river' enthält die zu schreibenden Flüsse als ein String. **Nicht maintainable! Dies sollte schnellstmöglich zu einem vernünftigen JSON-Format geändert werden!** Das 'authorization'-Feld enthält den Authentifizierungs-Token. Nur wenn dieser den User als einen Spielleiter identifiziert, wird diese Anfrage akzeptiert.
Die Anfrage liefert Statuscodes mit folgender Bedeutung zurück:

- 200: Erfolg
- 401: Nutzer nicht eingeloggt
- 403: Eingeloggter Nutzer hat keine SL-Rechte

### 2.5 Grenzen

Die Gebäude werden mit der Anfrage an "/databaseLink/saveborderdata/" gespeichert. Der Inhalt der Anfrage muss wie folgt aussehen:

```
{
  'borders': [{
    'tag': Kontrolierendes Reich (Kürzel),
    'land': [
      [x-Koordinate, y-Koordinate], ...
    ]
  }, ... ],
  'authorization': Auth-Token
}
```

Das Feld 'borders' enthält die zu schreibenden Grenzen im selben Format wie es vom Server empfangen wurde. Sie müssen mit der Stringify-Methode zu einem String verpackt werden. Das 'authorization'-Feld enthält den Authentifizierungs-Token. Nur wenn dieser den User als einen Spielleiter identifiziert, wird diese Anfrage akzeptiert.
Die Anfrage liefert Statuscodes mit folgender Bedeutung zurück:

- 200: Erfolg
- 401: Nutzer nicht eingeloggt
- 403: Eingeloggter Nutzer hat keine SL-Rechte

## 3 Sicherheit

Hier beschreiben wir das Sicherheitskonzept unserer Anwendung und die dazu relevanten Serverinteraktionen. Das generelle Sicherheitsdesign findet man im Unterabschnitt 3.1. Die Login-Interaktion ist beschrieben in 3.2 und der dabei benutzte CSRF-Token in 3.3. Der Logout ist im Unterabschnitt 3.4 erklärt.

### 3.1 Sicherheitsdesign

Das Phoenixtool setzt alle gängigen Sicherheitsvorkehrungen für Webanwendungen ein, um die Integrität des Spiels sicher zu stellen.

Die Kommunikation mit dem Server erfolgt über das HTTPS Protokoll. Ein User muss sich beim Server einloggen um nicht-öffentliche Information zu erhalten und damit seine Anfragen vom Server akzeptiert werden.
Der Login erfolgt durch die vom Django-Framework bereitgestellte Infrastruktur und ein eingeloggter User erhält ein Authentifizierungs-Token (authentification token). Dieser muss bei bestimmten Anfragen mitgeschickt werden und der Server überprüft, ob der User für die gegebene Anfrage autorisiert ist. Zum Beispiel ist nur der Spielleiter und der Besitzer einer Armee autorisiert, diese zu bewegen.
Um redundante Regelimplementierung zu vermeiden, führt der Server nur die rudimentäre Berechtigungsüberprüfung durch. Die Regeln sind nur in der Client-Anwendung implementiert. Da wir die Integrität der Client-Software nicht sicherstellen können, implementieren wir den Event-Sicherheitsmechanismus. Dieser verhindert, dass ein kompromittierter Spieler-Client die Integrität des gesamten Spieles kompromittiert. Zusätzlich erlaubt es der Spielleitung mehr und feinere Kontrolle über den Spielverlauf.
Der Event-Mechanismus verbietet Spielern direkt Änderungen an dem serverseitigen Spielstand zu machen. Stattdessen schreiben sie so genannte "Events" in die Server-Datenbank, welche die Absicht repräsentieren, eine bestimmte Aktion durchzuführen. Diese lädt die Spielleitung dann in ihre Client-Software und transformiert diese dann manuell oder (teil)automatisch in tatsächliche Spielstandänderung.
Die Client-Software der Spielleitung ist die selbe Implementierung wie die der Spieler, weshalb immer noch keine redundante Regelimplementierung vorliegt. Die unterschiedlichen Möglichkeiten der Spieler- und SL-Clients beruht ausschließlich auf die unterschiedlichen Authentifizierungstokens und die entsprechende Reaktion des Servers auf die generierte Anfragen. Anderes als bei dem Spieler-Client wird angenommen, dass alle Veränderungen, welche die SL an ihrem Client vornimmt, legitim sind.
Die Event-Konstrukte und deren Benutzung sind in den Abschnitten 4.3 und 5 beschrieben.

## 3.2 Login

Mit einer Anfrage an "/databaseLink/login/" loggt man sich auf dem Server ein. Die Anfrage muss wie folgt aussehen:

```
{
  'username': Nutzername,
  'password': Passwort
}
```

Die Antwort kommt in folgendem Format:

```
{
  'token': Auth-Token,
  'group': 'guest'/'sl'/Reichskürzel
}
```

Das 'token'-Feld enthält den Authentifizierungstoken, welchen man braucht, um sich bei anderen Anfragen zu authentifizieren. Das 'group'-Feld enthält einen Text, der beschreibt, als welche Nutzergruppe (bestimmtes Reich, Spielleitung oder Nutzer ohne irgendwelche Rechte) man sich eingeloggt hat. Dies ist nur dazu da, um eine einfach lesbare Bestätigung zu haben, als welcher Nutzer man sich eingeloggt hat. Als fälschungssichere Bestätigung der Nutzeridentität dient das Authentifizierungstoken.
***Es fehlen mehrere wichtige Sicherheitsmechanismen, die möglichst bald implementiert werden sollen.***

1. ***Zum Schutz vor Cross Site Request Forgery soll ein CSRF-Token benutzt werden***
2. ***Die Kommunikation mit dem Server soll auf HTTPS umgestellt werden, um man-in-the-middle Angriffe weniger mächtig zu machen***

### 3.3 CSRF Token

***Muss noch implementiert werden.***

### 3.4 Logout

Um sich auszuloggen muss man eine Anfrage an "/databaseLink/logout/" schicken.

## 4 Zugstruktur

In diesem Abschnitt ist erklärt, wie die Zugabfolge des Phönix-Spiels in unserer Software serverseitig umgesetzt ist. Das Auslesen des aktuellen Zuges und die Aktionen, in jedem Zugabschnitt erwartet sind, beschreiben wir in 4.1. Wie man den nächsten Zugabschnitt auslöst ist in 4.2 erklärt. Das System der Events, welche Teil unseres Sicherheitskonzepts sind, werden in der Zugstruktur benutzt. Die Einzelheiten dieser Nutzung sind in 4.3 beschrieben.

### 4.1 Aktueller Zug

Die grundsätzliche Zugstruktur jede Runde sieht wie folgt aus:

- 0. Zug 147. kein Reich
- 1. Zug 147. Reich 1 Start
- 1. Zug 147. Reich 1 Ende
- 2. Zug 147. Reich 2 Start
- 2. Zug 147. Reich 2 Ende
- ...
- n. Zug 147. Reich N Start
- n. Zug 147. Reich N Ende
- 0. Zug 148. kein Reich
- ...

Am Anfang jedes Zuges ist die SL ein mal dran um alles, was am Anfang des Zuges durchgeführt werden muss, zu machen. Dazu gehört das Auffüllen von Gutpunkten, Baupunkten und Einkommen. Hier sind auch die Spieler in Rüstrunden aktionsberechtingt. Dann ist das erste Reich dran um ihren Zug in Form von Events abzugeben. Anschließend ist sie Spielleitung dran und bearbeitet die abgegebenen Events. Wenn die SL ihren Zwischenzug abschließt, beginnt der Zug des nächsten Reiches. Das geht so lange weiter bis alle Reiche ein mal dran waren. In ihrer letzten Zwischenzugphase muss die SL neben dem Abhandeln des Zuges des letzten Reiches alles durchführen, was am Ende eines Zuges passieren muss.
Den aktuellen Zug bekommt man mit der Anfrage "/databaseLink/getturn/". Die Antwort kommt in folgendem Format:

```
{
  'turn': Zugnummer (z.B. 147),
  'realm': Reich am Zug (Kürzel) oder 'sl',
  'status': 'st' oder 'fi'
}
```

Wenn das 'realm'-Feld 'sl' beinhaltet, ist gerade die Zug-Anfangsphase und kein Reich ist dran. Bei 'status' steht 'st' für Start und 'fi' für Ende/Finish.

### 4.2 Nächster Zug

Man beendet eine Zugphase und startet die nächste mit einer Anfrage an "/databaseLink/-nextturn/". Der Inhalt ist

```
{'authorization': Auth-Token}
```

Die Anfrage gibt, wenn angenommen, den neuen aktuellen Zug in dem oben (4.1) erklärten Format zurück. Ein Spieler kann nur den aktiven (Status 'st') seines Reiches beenden und den bis auf den Status gleichen Zug anfangen. Ein SL-User kann die Zug-Zwischenphasen (Status 'fi') und die Runden-Anfangsphase (Reich 'sl') beenden und den Zug des nächsten Reiches beginnen. Die SL

kann auch die Zugphase eines aktiven Reiches beenden. Dies sollte nur in absoluten Ausnahmefällen geschehen und die Client-Software sollte eine entsprechende Warnung anzeigen, um Fehler seitens der SL zu vermeiden.
Die Anfrage liefert Statuscodes mit folgender Bedeutung zurück:

- 401: Nutzer nicht eingeloggt
- 403: Der eingeloggte Nutzer ist nicht SL oder Mitglied des Reiches, welches am Zug ist
- 520: Die Datenbank hat keine Einträge für die Zugreihenfolge (wenn der eingeloggte Nutzer nicht SL ist)
- 521: Die Datenbank hat keine Einträge für die Zugreihenfolge (wenn der eingeloggte Nutzer SL ist)

## 4.3 Nutzung von Events

Es gibt verschiedenen Event-Typen, die auf dem Server getrennt gespeichert werden. Alle Events werden von dem Server in folgendem Format bereit gestellt:

```
{
  'type': 'move'/'battle'/...,
  'content': {},
  'pk': value
}
```

Der Einhalt des Events ist im Feld 'content' und unterscheidet sich je nach Typ des Events. Der Typ des Events wird im Feld 'type' als Text bereitgestellt. Das letzte Feld 'pk' enthält den Private-Key, welchen das Event in der Server-Datenbank hat. Dieser muss zur Verwendung von Events zurück an den Server geschickt werden, damit das richtige Event identifiziert werden kann.
Die Benutzung der Events beinhaltet vier Schritte: neue Events anlegen (beschrieben in 5), die nicht-bearbeiteten Events holen (beschrieben in 4.3.1), Events bearbeiten (beschrieben in 4.3.2) und Events löschen (beschrieben in 4.3.3). Diese sind nachfolgend beschrieben.
Nur die Spielleitung kann Events bearbeiten und löschen. Neue Events können sowohl die Spieler als auch die SL anlegen, wobei es im normalen Betrieb nicht vorgesehen ist, dass die SL neue Events anlegt. Die Events sind nur dazu gedacht, eine Trennschicht zwischen den Spieler-Clients und dem Spielstand in der Datenbank zu schaffen. Für die SL ist diese Sicherheitstrennung nicht notwendig, da die von ihnen erzeugte Spielstände als trivial korrekt angenommen wird.

### 4.3.1 Ausstehende Events Holen

Neu erzeugte Events sind auf dem Server intern als ausstehend markiert. Das bedeutet, dass sie noch nicht von der Spielleitung bearbeitet wurden.
Wenn die SL ihre Zwischenzugphase startet, muss ihre Client-Software zuerst alle ausstehende Events holen. Dazu muss eine Anfrage an die URL "/databaseLink/getevents/" geschickt werden. Als Antwort erhält man die Events als JSON-Array mit Elementen im in 4.3 beschriebenen Format.

### 4.3.2 Events Bearbeiten

Wenn die SL ein Event bearbeitet, muss sie die passende Spielstandänderung durchführen und auf den Server schreiben (beschrieben in 2) und das Event auf dem Server als bearbeitet markieren lassen. Dazu schickt man eine Anfrage an "/databaseLink/checkevent/". Die Anfrage muss folgendes Format haben:

```
{
  'authorization': Auth-Token,
  'eventId': 'pk'-Feld,
  'eventType': 'type'-Feld
}
```

Die Anfrage wird abgelehnt, wenn der Authentifizierungs-Token nicht einen einge-

loggten SL-User ausweist. Nach dem ein Event als bearbeitet markiert wurde, bleibt es in der Datenbank statt gelöscht zu werden. So kann man aus bearbeiteten Events die Historie des Spielverlaufs rekonstruieren.
Die Anfrage liefert Statuscodes mit folgender Bedeutung zurück:

- 200: Erfolg
- 403: Der eingeloggte Nutzer ist nicht SL

#### 4.3.3 Events Löschen

Wenn die SL ein Event als nicht legitim erachtet, muss sie es vom Server löschen. Man beachte, dass nur fehlerhafte und ungültige Events gelöscht werden sollen. Rechtmäßige, bearbeitete Events sollen, wie in 4.3.2 beschreiben, auf bearbeitet gestellt werden und in der Datenbank verblieben.
Um ein Event zu löschen, muss man eine Anfrage an "/databaseLink/deleteevent/" schicken. Genauso wie bei der Event-Bearbeitung muss die Anfrage wie folgt aussehen:

```
{
  'authorization': Auth-Token,
  'eventId': 'pk'-Feld,
  'eventType': 'type'-Feld
}
```

Die Anfrage wird nur akzeptiert, wenn sie von einem eingeloggten SL-User stammt (durch den Authentifizierungs-Token ausgewiesen).
Die Anfrage liefert Statuscodes mit folgender Bedeutung zurück:

- 200: Erfolg
- 403: Der eingeloggte Nutzer ist nicht SL

## 5 Events

Dieser Abschnitt beschreibt den Inhalt einzelner Event-Typen. Der für die Truppenbewegung repräsentierende Move-Event ist in 5.1 erklärt. Die Battle-Events, die (potentielle) Schlachten zwischen verschiedenen Armeen repräsentieren, sind in 5.2 beschrieben.

### 5.1 Move Event

Ein Move-Event beschreibt die Bewegung einer Armee oder Flotte ein Feld weit (Kleinfeld im Erkenfara-Regelwerk). Eine Armee/Flotte kann sich unter Umständen mehrere Felder pro Zug bewegen. In diesem Fall wird die Bewegung durch mehrere Move-Events dargestellt. Das 'content'-Feld eines Move-Events hat folgendem Aufbau:

```
{
  'armyId': Armeenummer (Phoenix),
  'realm': Reich der Armee (Kürzel),
  'fromX': x-Koordinate Herkunft,
  'fromY': y-Koordinate Herkunft,
  'toX': x-Koordinate Ziel,
  'toY': y-Koordinate Ziel
}
```

Ein neues Event fügt man mit einer Anfrage auf "/databaseLink/moveevent/" hinzu. Die Anfrage sieht wie folgt aus:

```
{
  'authorization': Auth-Token,
  'content': {wie oben}
}
```

Alle vom eingeloggten SL-User geschickten Move-Events werden in die Datenbank aufgenommen. Bei Spielern überprüft der Server zuerst, ob der Spieler zum Reich der zu bewegenden Truppe gehört.
Die Anfrage liefert Statuscodes mit folgender Bedeutung zurück:

- 200: Erfolg
- 400: Das zu bewegende Heer existiert nicht
- 401: Nutzer ist nicht eingeloggt
- 403: Der eingeloggte Nutzer ist nicht Besitzer des Heeres oder SL

### 5.2 Battle Event

Ein Battle-Event beschreibt eine potentielle Schlacht zwischen mehreren Armeen/Flotten. Das 'content'-Feld hat den folgenden Inhalt:

```
{
  'x': x-Koordinate Schlachtfeld,
  'y': y-Koordinate Schlachtfeld,
  'participants': [
    {
      'armyId': Armeenummer (Phoenix),
      'realm': Reich der Armee (Kürzel)
    },
    ...
  ]
}
```

Das 'participants'-Array ist die Liste aller potentiell an der Schlacht teilnehmender Armeen. Es sind also alle Armeen (bzw. Flotten), die sich zum Zeitpunkt der Schlacht auf dem Feld aufhalten werden. Falls Armeen auf dem Feld sind, die nicht an der Schlacht teilnehmen sollen (z.B. wenn eine dritte, neutrale Partei auf dem Feld ist), kann es bei der Auswertung berücksichtigt werden und muss nicht in der Datenbank repräsentiert sein. Genauso wenig sind die potentiellen Teilnehmer nicht nach Seiten sortiert.
Ein neues Event fügt man mit einer Anfrage auf "/databaseLink/battleevent/" hinzu. Die Anfrage sieht wie folgt aus:

```
{
  'authorization': Auth-Token,
  'content': {wie oben}
}
```

Alle von der SL geschickten Battle-Events werden in die Datenbank aufgenommen. Bei Spielern muss mindestens eine der an der Schlacht teilnehmenden Armeen/Flotten zum Reich des Spielers gehören, damit das Event akzeptiert wird.
Die Anfrage liefert Statuscodes mit folgender Bedeutung zurück:

- 200: Erfolg
- 400: Das zu bewegende Heer existiert nicht
- 401: Nutzer ist nicht eingeloggt
- 403: Der eingeloggte Nutzer ist nicht SL oder Besitzer von mindestens einem Heer, welches an der Schlacht teilnimmt

### 5.3 Shoot Event

Ein Shoot-Event beschreibt einen Fernkampf einer Armee auf ein Zielfeld. Das 'content'-Feld hat den folgenden Inhalt:

```
{
  'armyId': Armeenummer (Phoenix),
  'realm': Reich der Armee (Kürzel),
  'LKPcount': Anzahl an leichten Katapulten,
  'SKPcount': Anzahl an schweren Katapulten,
  'toX': x-Koordinate Ziel,
  'toY': y-Koordinate Ziel
}
```

Ein neues Event fügt man mit einer Anfrage auf "/databaseLink/shootevent/" hinzu. Die Anfrage sieht wie folgt aus:

```
{
  'authorization': Auth-Token,
  'content': {wie oben}
}
```

Alle vom eingeloggten SL-User geschickten Shoot-Events werden in die Datenbank aufgenommen. Bei Spielern überprüft der Server zuerst, ob der Spieler zum Reich der schießenden Truppe gehört.
Die Anfrage liefert Statuscodes mit folgender Bedeutung zurück:

- 200: Erfolg
- 400: Das schießende Heer existiert nicht
- 401: Nutzer ist nicht eingeloggt
- 403: Der eingeloggte Nutzer ist nicht SL oder Besitzer des Heeres das shießen möchte.