Standard-FantasticFeasts-REVIEWME-14

Standardisierungsdokument zum Softwaregrundprojekt 18/19

sopra teams

18. Mai 2019

Contents

1 Changelog

Um die genauen Änderungung zu verfolgen bitte `git diff` verwenden. Changelog könnte unvollständig sein.

1.1 Changelog REVIEWME-13 -> REVIEWME-14

- [x] Wombat

- [x] neue phaseType `unbanPhase`
- [x] deltas weiter beschrieben
- [x] `goalWasThrownThisRound` flag in snapshot
- [x] Error und Warnings weiter gearbeit und verpflichtend gemacht
- [x] Falls ein Spieler einen Quaffel hält wird dieser nicht mit teleportiert.
- [x] commandline args (optional) Name der Lobby (default –lobby hogwarts)
- [x] foolAway Wahrscheinlichkeit aus matchConfig entfernt
- [x] Annmerkung: das message feld bei der Pause kann frei ausgefüllt werden
- [x] Teams können eigene Error und Warning Codes definieren
- [x] neue Felder wegen unban in matchConfig
- [x] Torfelder sind keine gültingen Startpositionen.
- [x] bei Chatmod reconnectChat hinzugefügt

1.2 Changelog REVIEWME-12 -> REVIEWME-13

- [x] clarification of "Next" desciption
- [x] neue Zeicheneinschränkung `(string/motto)` in teamConfig fuer Motto
- [x] unban auch als Broadcast
- [x] Definition fuer skip ausgebessert, Quaffel nicht mehr müde
- [x] Tabelle mit Broadcats/Unicasts korrigiert
- [x] Artefaktliste
- [x] einige genauere Beschreibungen und Verbesserungen
- [x] Broadcast werden nur in Lobby gesendet angemerkt
- [x] `matchConfig` verändert. `timeouts` -> unbenannt -> `timings` (AnimationDuration)
- [x] Felder in `timings` umbenannt
- [x] `"teamFormationTimeout": "(int/millisec)"` in matchConfig hinzugefügt
- [x] Hexadezimaler RGB-Wert in GROßBUCHSTABEN
- [x] Wahrscheinlichkeit als (Komma-)Zahl von 0 bis 1 inkludiert
- [x] Zuerst schickt der Server `joinResponse` dannach `loginGreeting`. (mit Ablauf)
- [x] neuer deltaType `turnUsed`
- [x] fans haben ein `"turnUsed" : "(boolean)"` Feld im Snapshot
- [x] Reconnect Example
- [x] `(string/broom)` jetzt einheitlich ohne Bindestriche
- [x] `(int/maxRounds)` definiert 13 <= maxRounds <= 100
- [x] Kommandozeilenparameter lesbarer gemacht
- [x] Der erste Snapshot beinhaltet als lastDeltaBroadcast einen `roundChange`.
- [x] neues Feld im delta `(string/banReason)`
- [x] Bessere PDF Usability durch Hyperlinks
- [x] Bei verspätet eintreffenden Deltas werden diese ignoriert und der Server kickt den Client nicht!
- [x] Protokollverletzung definiert

1.3 Changelog REVIEWME-11 -> REVIEWME-12

- [x] neue Zeicheneinschränkung `(string/name)` in teamConfig für Teamname und Namen.
- [x] neuer deltaType foolAway
- [x] deltas genauer zwischen deltaRequest und deltaBroadcast unterschieden
- [x] die payloadType deltaUnicast und nextUnicast entfernt
- [x] reconnect neues Feld next
- [x] neue deltaTypes `phaseChange`, `goalPointsChange`, `roundChange`, `skip`, `unban`

- [x] neue felder in deltas
- [x] container im reconnect
- [x] Es werden keine `deltaBroadcast` mehr verschickt. Deltas befinden sich jetzt innerhalb eines snapshots.

1.4 Changelog REVIEWME-10 -> REVIEWME-11

- [x] Im Beispiel “success”: “false” -> verbessert -> “success”: false
- [x] Im Beispiel timeout“:”7500" -> verbessert -> “timeout”:7500
- [x] password gehört zum userName (`joinRequest`)
- [x] Reihenfolge in der Json egal angemerkt.
- [x] Eine Partie gleichzeitig pro lobby. Server muss mindestens eine lobby unterstützen.
- [x] Der Server kickt den Client bei falsch eintreffenden `deltaRequest`
- [x] Optionen fuer das Lobbyfeld genauer definiert `(string/lobbyType)`
- [x] Das Replay ist erst nach dem MatchFinish anforderbar
- [x] In einer Lobby können hintereinander mehre Spiele gespielt werden.
- [x] success Feld in deltas genauer beschrieben
- [x] `knockout` feld im Snapshot
- [x] default lobby ist `hogwarts`
- [x] Ein Beispiel zum ban und neuer turnType removeBan
- [x] Error and Warning List
- [x] container Example

1.5 Changelog REVIEWME-9 -> REVIEWME-10

- [x] Der Login stellt Identitätsabsicherung
- [x] payloadType in Definition vergessen.
- [x] Alle Json Nachrichten werden per Websocket Protokoll verschickt. (https://tools.ietf.org/html/rfc6455)
- [x] Im `replay` timestamp statt date und startTimestamp Type
- [x] Der neuste Login zählt.
- [x] userName für den gesamten Server
- [x] alphanumerisch -> unbenannt -> userNameType
- [x] userNameType Zeichenlänge definiert
- [x] username und password werden in der KI hardgecoded
- [x] `null` falls der Schnatz noch nicht auf dem Spielfeld ist

1.6 Changelog REVIEWME-8 -> REVIEWME-9

- [x] “matchConfig” und “teamConfig” als File auch im Container Format präzisiert.
- [x] in Next `(int/millisec)`
- [x] Die Farben `primary` und `secondary` müssen unterschiedlich sein.
- [x] teamFormation startpositionen an pflichtenheft angepasst (appendix).

1.7 Changelog REVIEWME-7 -> REVIEWME-8

- [x] elfs -> unbenannt -> elves
- [x] Changelog in Dokument übernommen
- [x] In deltas müssen `null`-Felder mitgeschickt werden
- [x] neuer `deltaType` wrestQuaffle

- [x] Fehler in Type Declaration z.B. `(int/ballPhase)`-> unbenannt ->`ballPhase`
- [x] delta json formatierung angepasst
- [x] `(string/fan)` gelöschen da es definiert war aber nicht benutzt wurde

1.8 Changelog REVIEWME-4 -> REVIEWME-7

- [x] (int/Difficulty) -> unbenannt -> (int/difficulty)
- [x] (int/Verbosity) -> unbenannt -> (int/verbosity)
- [x] (int/actionPhase) gelöschen
- [x] turnType fan hinzufügen
- [x] Beschreibung das man timeout in `next` braucht wegen Pause
- [x] Ersichtlich beschreiben das reconnect für Zuschauer und Spieler gleich abläuft
- [x] Im Login, Server muss die Verbindung schließen wenn er gewünschte mods nicht unterstützt
- [x] kurz Spielablauf beschrieben am Anfang des Dokumentes
- [x] remove propabilities goal aus matchConfig
- [x] propability -> unbenannt -> probability
- [x] thinderblast -> unbenannt -> tinderblast
- [x] passiveEntity in Beispiel und Beschreibung unterschiedlich ausgebessert
- [x] neue payloadType deltaUnicast und nextUnicast
- [x] Server muss beim Reconnect deltaUnicast und nextUnicast payloads schicken.
- [x] quaffleThrow Definition präzisiert.
- [x] teamFormation startpositionen präzisiert (appendix).

2 About this document

Dieses Dokument soll ein strukturiertes Nachschlagewerk sein um beim Implementieren der jeweiligen Komponenten einen roten Faden zu haben. Alle Json Nachrichten werden per Websocket Protokoll verschickt. (https://tools.ietf.org/html/rfc6455)

Das Protokoll ist delta und snapshot basiert. Bei Protokollverletzung wird der Client gekickt. Falls ein json nicht im Container Format geschickt wird, ist es eine Protokollverletzung. Des Weiteren wenn die vorgegeben Typen über ihren definierten Bereich hinausgehen. (z.B. Wahrscheinlichkeit > 1, nicht spezifizierte Besentypen, falsche `teamConfig`) ... Alles was der Spiellogik aus dem Lastenheft widerspricht, ist eine Protokollverletzung folglich wird der Client gekickt. z.B. Spieler versucht mehr als ein Entfernung zu bewegen, Spieler versucht sich außerhalb des Spielfeld zu bewegen. Klatscher wird mehr als 3 Entfernung gekloppt... Es ist egal in welcher Reihenfolge die einzelnen Felder in der Json stehen.

2.1 Whole Game Sequence in short

Erst einmal interessiert einen der Verlauf des Spiels. Hier ist in sehr kurzer Weise dargestellt wie ein Spiel abläuft. Einigen wird schon auffallen das es sehr viele Fälle gibt die noch nicht abgedeckt sind. Deswegen sind genauere Beschreibungen in den betreffenden Abschnitten behandelt. Auch sind hier die Formate welche für die Züge der Spieler verantwortlich sind nicht beschrieben. Es ist mehr ein Ablauf welcher von einem Zuschauer gesehen wird, wobei auch dieser mehr vom Spiel mitbekommen wird.

- `joinRequest`: Ist in jedem Fall die erste Nachricht, welche ein Client an den Server sendet. Bei einem erfolgreichen Login befindet man sich in einer Lobby und ist vorerst ein Zuschauer.

- `teamConfig`: Ein Zuschauer wird zum Spieler, indem er seine Team-Konfiguration an den Server sendet und diese erfolgreich angenommen wird.
- `matchStart`: Das Spiel startet, sobald es zwei Spieler gibt. Teilt beiden Spielern die Spielfeldseite zu.
- `teamFormation`: Beide Spieler stellen alle ihre Spielfiguren auf die linke bzw. rechte Spielfeldseite auf.
- `snapshot`: Ein Snapshot ist ein valider Zustand des Spiels. Jeder Snapshot wird vom Server erzeugt und anschließend an alle Clients verteilt. Der erste Snapshot wird gesendet, nachdem beide Teams aufgestellt sind, als Bestätigung für beide Teamaufstellungen. Snapshots enthalten den letzten deltaBroadcast. Es werden keine Deltas einzelnt verschickt.
- `matchFinish`: Markiert das Ende des Spiels.

2.1.1 Definition Sequence

Damit einfacher der sequenzielle Ablauf definiert werden kann. Ist hier folgende Schreibweise und Bedeutung eingeführt.

- request: Client sendet an Server
- broadcast: Server sendet an alle, Spieler und Zuschauer
- unicast: Server sendet an einen Client
- Der Spielverlauf aus Sicht eines Servers. D.h. alle Verbindungen werden geloggt.
- (... markiert ausgelassene Nachrichten)

2.1.1.1 Example

```
# server start
...
request  : joinRequest (peter meldet sich an)
...
request  : joinRequest (sandra meldet sich an)
...
request  : joinRequest (norbert meldet sich an)
...
request  : teamConfig (norbert sendet seine teamConfig)
request  : teamConfig (sandra sendet ihre teamConfig)
broadcast: matchStart (norbert ist linker Spieler, sandra ist rechter Spieler)
request  : teamFormation (sandra stellt ihr Team auf)
request  : teamFormation (norbert stellt sein Team auf)
broadcast: snapshot (initialer Snapshot)
broadcast: snapshot (Klatscher1 wurde bewegt)
broadcast: snapshot (Klatscher2 wurde bewegt)
...
broadcast: snapshot (neuer Snapshot mit neuer Spielerposition)
...
broadcast: snapshot (letzter Snapshot)
broadcast: matchFinish (Spiel ist zu Ende und sandra hat gewonnen)
...
# server end
```

3 Kommandozeilenparameter für KI und Server

3.1 Type Declaration

3.1.1 `(int/difficulty)`

- 0: Maximale Schwierigkeit
- 1: weniger schwer
- 2: noch weniger schwer
- 3: ...
- n: naiv

3.1.2 `(int/verbosity)`

- 0: keine Ausgabe
- 1: mehr Ausgaben
- 2: noch mehr Ausgaben
- 3: ...
- n: alles

3.2 Definition

- `(optional)` muss implementiert werden, nicht unbedingt zum starten benötigt.

3.2.1 KI

```
usage: ki-teamXX
 -a --address <host>     Adresse des Servers
 -t --team <path>        Pfad zur Team-Konfiguration
 -l --lobby <string>     (optional) Name der Lobby (default --lobby hogwarts)
 -u --username <string>  (optional) userName der KI
 -k --password <string>  (optional) Passwort der KI
 -h --help               (optional) Hilfe
 -p --port <port>        (optional) Server-Port (default --port 4488)
 -d --difficulty <int>   (optional) Schwierigkeit
 -v --verbosity <int>    (optional) Verbosity (Ausführlichkeit der Log-Information)
```

3.2.2 Server

```
usage: server-teamXX
 -m --match <path>     Pfad zur Partie-Konfiguration
 -h --help             (optional) Hilfe
 -p --port <port>      (optional) Server-Port
 -v --verbosity <int>  (optional) Verbosity (ausführlichkeit der Log-Information)
```

4 Container

4.1 Description

Der Container ist ein valides Json-Objekt und beschreibt den strukturellen Zusammenhang der definierenten jsonpayload Formate. Falls der payloadType hier nicht definiert ist, wird er einfach ignoriert. Falls es Felder im json gibt, welche hier nicht definiert sind, werden sie ebenfalls ignoriert. Der timestamp welcher vom Client geschickt wurde muss nicht richtig sein und wird vom Server mit der Ankunftszeit des Tcp-packets überschrieben.

4.2 Type Declaration

4.2.1 `(string/payloadType)`

- `joinRequest`
- `loginGreeting`
- `joinResponse`
- `sendDebug`
- `globalDebug`
- `privateDebug`
- `matchConfig`
- `teamConfig`
- `matchStart`
- `teamFormation`
- `snapshot`
- `pauseRequest`
- `continueRequest`
- `pauseResponse`
- `next`
- `deltaRequest`
- `deltaBroadcast`
- `matchFinish`
- `getReplay`
- `replay`
- `reconnect`

4.3 Definition

4.3.1 container

```
{
  "timestamp": "(string/yyyy-MM-dd HH:mm:ss.SSS)",
  "payloadType" : "(string/payloadType)",
  "payload": "(json)"
}
```

4.3.2 Example

Alle Websocket-Messages befinden sich im Container-Format.

```
{
  "timestamp": "2019-04-18 17:20:12.111",
  "payloadType": "examplepayloadTypethatdoentexist",
  "payload": {
    "that": "is",
    "only": [
      "an",
      "example"
    ],
    "everything": "will",
    "be": 8,
    "json": true
  }
}
```

4.3.3 Overview

- request: Client sendet an Server
- broadcast: Server sendet an alle (in der Lobby), Spieler und Zuschauer
- unicast: Server sendet an einen Client

request	broadcast	unicast	siehe
joinRequest	loginGreeting	joinResponse	siehe Login
sendDebug	globalDebug	privateDebug	siehe Debug
teamConfig			siehe TeamConfig
	matchStart		siehe MatchStart
teamFormation			siehe TeamFormation
	snapshot		siehe Snapshot
pauseRequest, continueRequest	pauseResponse		siehe Pause
	next		siehe Next
deltaRequest			siehe Delta
	matchFinish		siehe MatchFinish
getReplay	replay		siehe Replay
		reconnect	siehe Reconnect
		privateError	siehe Error and Warning
		privateWarning	siehe Error and Warning

4.3.4 Liste von Artefakten (werden im Container-Format persistent abgespeichert)

- teamConfig (siehe TeamConfig)
- matchConfig (siehe MatchConfig)
- replay (siehe Replay)

4.4 Debug

4.4.1 Description

Server und Client können jederzeit debug Information senden. Verhalten ist nicht weiter definiert, d.h. Ignorieren, sich mit anderen unterhalten, Fehlermeldungen an den Client senden, Shakespeare

vorlesen. Der Server sollte keine unnötigen Debug-Informationen an Clients senden.

4.4.2 Definition

4.4.2.1 sendDebug

```
{
  "information" : "(string)"
}
```

4.4.2.2 globalDebug

```
{
  "information" : "(string)"
}
```

4.4.2.3 privateDebug

```
{
  "information" : "(string)"
}
```

4.4.3 Example

```
{
  "timestamp": "2019-02-11 11:11:11.111",
  "payloadType" : "sendDebug",
  "payload": {
    "information" : "hagrid, I love Magic"
  }
}
```

```
{
  "timestamp": "2019-02-11 11:13:12.111",
  "payloadType" : "globalDebug",
  "payload": {
    "information" : "50 points to gryffindor"
  }
}
```

```
{
  "timestamp": "2019-02-11 12:10:12.111",
  "payloadType" : "privateDebug",
  "payload": {
    "information" : "you are wizard harry"
  }
}
```

4.5 Login

Die `joinRequest` ist in jedem Fall die erste Nachricht, welche ein Client an den Server sendet. Bei einem erfolgreichen Login befindet man sich in einer Lobby und ist vorerst ein Zuschauer. Alle Broadcast werden nur in diese Lobby gesendet. Falls der Server die angegebenen mods nicht unterstützt, muss die Verbindung vom Server geschlossen werden. Der Login stellt Identitätsabsicherung. Der neuste Login zählt. Eine alte Verbindung muss vom Server geschlossen werden. Es kann immer nur eine Verbindung für einen userName existieren. Das password gehört zum userName. Der Server speichert den userName plus password für den gesamten Server. Der Server unterstuetzen immer nur eine Partie gleichzeitig pro lobby. Ein Server muss mindestens eine lobby zugänglich haben. Die default lobby ist `hogwarts`. Zuerst schickt der Server `joinResponse` dannach `loginGreeting`.

4.5.1 Type Declaration

4.5.1.1 `(string/userNameType)`

- Darf nur folgende Zeichen enthalten `regex=[a-zA-Z0-9]`
- muss 3 bis 20 Zeichen lang sein. (Anmerkung Zeichen != Byte)

4.5.1.2 `(string/lobbyType)`

- Darf nur folgende Zeichen enthalten `regex=[a-zA-Z0-9]`
- muss 3 bis 40 Zeichen lang sein. (Anmerkung Zeichen != Byte)

4.5.2 Definition

4.5.2.1 joinRequest

```
{
  "lobby": "(string/lobbyType)",
  "userName": "(string/userNameType)",
  "password" : "(string)",
  "isArtificialIntelligence":"(boolean)",
  "mods":["(string)"]
}
```

4.5.2.2 loginGreeting

```
{
  "userName" : "(string/userNameType)"
}
```

4.5.2.3 joinResponse

```
{
  "message" : "(string)"
}
```

4.5.3 Example

```
{
  "timestamp": "2019-02-11 11:13:12.111",
  "payloadType" : "joinRequest",
  "payload": {
    "lobby": "MemesOnly",
    "userName": "dealwithit",
    "password" : "1337",
    "isArtificialIntelligence":false,
    "mods":[]
  }
}
```

```
{
  "timestamp": "2019-02-11 11:13:12.112",
  "payloadType" : "loginGreeting",
  "payload": {
    "userName" : "dealwithit"
  }
}
```

```
{
  "timestamp": "2019-02-11 11:13:12.113",
  "payloadType" : "joinResponse",
  "payload": {
    "message" : "welcome, please enjoy"
  }
}
```

```
# Login start
request  : joinRequest
unicast  : joinResponse
broadcast: loginGreeting
# Login end
```

5 Konfigurationen für Team und Partie

5.1 Description

Die “matchConfig” und “teamConfig” sind Artefakte welche in einem File gespeichert werden. Abgespeichert werden sie im auch Container-Format.

5.2 Type Declaration

5.2.1 `(string/name)`

- Darf nur folgende Zeichen enthalten `regex=[a-zA-Z0-9 ]`
- ja, da ist ein Space im Regex.
- muss 3 bis 40 Zeichen lang sein. (Anmerkung Zeichen != Byte)

5.2.2 (string/motto)

- Darf nur folgende Zeichen enthalten a-z, A-Z, 0-9, Leerzeichen sowie die Satzzeichen .:,;?!'-
- muss 3 bis 200 Zeichen lang sein.

5.2.3 (string/role)

- `chaser` (Jäger)
- `beater` (Treiber)
- `keeper` (Hüter)
- `seeker` (Sucher)

5.2.4 (string/sex)

- `m` (Männlich)
- `f` (Weiblich)

5.2.5 (string/broom)

- `tinderblast`
- `cleansweep11`
- `comet260`
- `nimbus2001`
- `firebolt`

5.2.6 (string/RRGGBB)

- Hexadezimaler RGB-Wert in GROßBUCHSTABEN
- Bsp.: `C80010`

5.2.7 (float/prob)

- Wahrscheinlichkeit als (Komma-)Zahl von 0 bis 1 inkludiert
- Bsp.: `0.87`

5.2.8 (string/base64/png)

- Eine PNG-Datei base64-kodiert als String 256x256 Pixel

5.2.9 (int/maxRounds)

- 13 <= maxRounds <= 100

5.3 Definition

5.3.1 matchConfig (file)

Der Mindestabstand zwischen Snapshots ist durch AnimationDuration gegeben.

```
{
  "maxRounds": "(int/maxRounds)",
  "timings": {
    "teamFormationTimeout": "(int/millisec)",
    "playerTurnTimeout": "(int/millisec)",
    "fanTurnTimeout": "(int/millisec)",
    "unbanTurnTimeout": "(int/millisec)",
    \\ minimum time between snapshots
    "minPlayerPhaseAnimationDuration": "(int/millisec)",
    "minFanPhaseAnimationDuration": "(int/millisec)",
    "minBallPhaseAnimationDuration": "(int/millisec)",
    "minUnbanPhaseAnimationDuration": "(int/millisec)"
  },
  "probabilities": {
    "throwSuccess": "(float/prob)",
    "knockOut": "(float/prob)",
    "catchSnitch": "(float/prob)",
    "catchQuaffle": "(float/prob)",
    "wrestQuaffle": "(float/prob)",
    "extraMove": {
      "tinderblast": "(float/prob)",
      "cleansweep11": "(float/prob)",
      "comet260": "(float/prob)",
      "nimbus2001": "(float/prob)",
      "firebolt": "(float/prob)"
    },
    "foulDetection": {
      "flacking": "(float/prob)",
      "haversacking": "(float/prob)",
      "stooging": "(float/prob)",
      "blatching": "(float/prob)",
      "snitchnip": "(float/prob)"
    },
    "fanFoulDetection": {
      "elfTeleportation": "(float/prob)",
      "goblinShock": "(float/prob)",
      "trollRoar": "(float/prob)",
      "snitchSnatch": "(float/prob)",
      "wombatPoo": "(float/prob)"
    }
  }
}
```

5.3.2 teamConfig (file)

Die Farben `primary` und `secondary` müssen unterschiedlich sein.

```
{
  "name": "(string/name)",
  "motto": "(string/motto)",
  "colors": {
    "primary": "(string/RRGGBB)",
    "secondary": "(string/RRGGBB)"
  },
  "image": "(string/base64/png)",
  "fans": {
    "goblins": "(int)",
    "trolls": "(int)",
    "elves": "(int)",
    "nifflers": "(int)",
    "wombats":"(int)"
  },
  "players": {
    "seeker": {
      "name": "(string/name)",
      "broom": "(string/broom)",
      "sex": "(string/sex)"
    },
    "keeper": {
      "name": "(string/name)",
      "broom": "(string/broom)",
      "sex": "(string/sex)"
    },
    "chaser1": {
      "name": "(string/name)",
      "broom": "(string/broom)",
      "sex": "(string/sex)"
    },
    "chaser2": {
      "name": "(string/name)",
      "broom": "(string/broom)",
      "sex": "(string/sex)"
    },
    "chaser3": {
      "name": "(string/name)",
      "broom": "(string/broom)",
      "sex": "(string/sex)"
    },
    "beater1": {
      "name": "(string/name)",
      "broom": "(string/broom)",
      "sex": "(string/sex)"
    },
    "beater2": {
      "name": "(string/name)",
```

```
      "broom": "(string/broom)",
      "sex": "(string/sex)"
    }
  }
}
```

6 Match-Start-Finish

6.1 Description

Wird am Anfang und am Ende der Partie gesendet. Hiermit wird festgelegt, wer “linkes” bzw. “rechtes” Team ist, indem die Configs bzw. die UserNames auf “left” bzw. “right” gematcht werden. In einer Lobby kann nachdem ein MatchFinish ausgeloest wurde wieder ein MatchStart ausgeloest werden, d.h. Alles geht wieder von vorne los.

6.2 Type Declaration

6.2.1 `(string/victoryReasonType)`

- `disqualification`
- `bothDisqualificationMostPoints`
- `bothDisqualificationPointsEqualSnitchCatch`
- `bothDisqualificationPointsEqualLastDisqualification`
- `mostPoints`
- `pointsEqualSnitchCatch`
- `violationOfProtocol`

6.3 Definition

6.3.1 matchStart

```
{
  "matchConfig":"(json/matchConfig)",
  "leftTeamConfig":"(json/teamConfig)",
  "rightTeamConfig":"(json/teamConfig)",
  "leftTeamUserName":"(string/userNameType)",
  "rightTeamUserName":"(string/userNameType)"
}
```

6.3.2 matchFinish

```
{
    "endRound": "(int)",
    "leftPoints":"(int)",
    "rightPoints":"(int)",
    "winnerUserName": "(string/userNameType)",
    "victoryReason": "(string/victoryReasonType)"
}
```

7 TeamFormation

7.1 Description

Die Teamformation ist am Anfang des Spiels. Die verbundenen Spieler rechts und links stellen ihre Spielfiguren entsprechend der Spielregeln auf. Torfelder sind keine gültingen Startpositionen. (siehe appendix)

7.2 Definition

7.2.1 teamFormation

```
{
  "players": {
    "seeker": {
      "xPos": "(int/posx)",
      "yPos": "(int/posy)"
    },
    "keeper": {
      "xPos": "(int/posx)",
      "yPos": "(int/posy)"
    },
    "chaser1": {
      "xPos": "(int/posx)",
      "yPos": "(int/posy)"
    },
    "chaser2": {
      "xPos": "(int/posx)",
      "yPos": "(int/posy)"
    },
    "chaser3": {
      "xPos": "(int/posx)",
      "yPos": "(int/posy)"
    },
    "beater1": {
      "xPos": "(int/posx)",
      "yPos": "(int/posy)"
    },
    "beater2": {
      "xPos": "(int/posx)",
      "yPos": "(int/posy)"
    }
  }
}
```

8 Snapshot

8.1 Description

Ein Snapshot ist ein valider Zustand des Spiels. Jeder Snapshot wird vom Server erzeugt und anschließend an alle Clients verteilt. Der erste Snapshot wird gesendet, nachdem beide Teams aufgestellt sind, als Bestätigung für beide Teamaufstellungen. Der erste Snapshot beinhaltet als lastDeltaBroadcast einen `roundChange`. Der Server muss Snapshot schicken. Ist für die Leute die Guis gestallten interessant. Der Snapshot allein reicht aus um ein Spiel als spectator zuzuschauen.

8.2 Type Declaration

8.2.1 `(string/fanType)`

- `goblin`
- `troll`
- `elf`
- `niffler`
- `wombat`

8.2.2 `(int/posx)`

- [0..16]

8.2.3 `(int/posy)`

- [0..12]

8.2.4 `(string/phaseType)`

- `ballPhase`
- `playerPhase`
- `fanPhase`
- `gameFinish`
- `unbanPhase`

8.3 Definition

8.3.1 teamSnapshot

```
{
  "points":"(int)",
  "fans": [
    {
      "fanType": "(string/fanType)",
      "banned": "(boolean)",
      "turnUsed" : "(boolean)"
```

```
    }
  ],
  "players": {
    "seeker": {
      "xPos": "(int/posx)",
      "yPos": "(int/posy)",
      "banned": "(boolean)",
      "turnUsed" : "(boolean)",
      "knockout" : "(boolean)"
    },
    "keeper": {
      "xPos": "(int/posx)",
      "yPos": "(int/posy)",
      "banned": "(boolean)",
      "holdsQuaffle" : "(boolean)",
      "turnUsed" : "(boolean)",
      "knockout" : "(boolean)"
    },
    "chaser1": {
      "xPos": "(int/posx)",
      "yPos": "(int/posy)",
      "banned": "(boolean)",
      "holdsQuaffle" : "(boolean)",
      "turnUsed" : "(boolean)",
      "knockout" : "(boolean)"
    },
    "chaser2": {
      "xPos": "(int/posx)",
      "yPos": "(int/posy)",
      "banned": "(boolean)",
      "holdsQuaffle" : "(boolean)",
      "turnUsed" : "(boolean)",
      "knockout" : "(boolean)"
    },
    "chaser3": {
      "xPos": "(int/posx)",
      "yPos": "(int/posy)",
      "banned": "(boolean)",
      "holdsQuaffle" : "(boolean)",
      "turnUsed" : "(boolean)",
      "knockout" : "(boolean)"
    },
    "beater1": {
      "xPos": "(int/posx)",
      "yPos": "(int/posy)",
      "banned": "(boolean)",
      "holdsBludger" : "(boolean)",
      "turnUsed" : "(boolean)",
      "knockout" : "(boolean)"
    },
    "beater2": {
```

```
      "xPos": "(int/posx)",
      "yPos": "(int/posy)",
      "banned": "(boolean)",
      "holdsBludger" : "(boolean)",
      "turnUsed" : "(boolean)",
      "knockout" : "(boolean)"
    }
  }
}
```

8.3.2 snapshot

```
{
  "lastDeltaBroadcast": "(json/deltaBroadcast)",
  "phase":"(string/phaseType)",
  "spectatorUserName": [
    "(string/userNameType)"
  ],
  "round":"(int)",
  "leftTeam":"(json/teamSnapshot)",
  "rightTeam":"(json/teamSnapshot)",
  "balls": {
    "snitch": {
      "xPos": "(int/posx)",
      "yPos": "(int/posy)"
    },
    "quaffle": {
      "xPos": "(int/posx)",
      "yPos": "(int/posy)"
    },
    "bludger1": {
      "xPos": "(int/posx)",
      "yPos": "(int/posy)"
    },
    "bludger2": {
      "xPos": "(int/posx)",
      "yPos": "(int/posy)"
    }
  },
  "wombatCubes":[
    {
      "xPos": "(int/posx)",
      "yPos": "(int/posy)"
    }
  ],
  "goalWasThrownThisRound": "(boolean)"
}
```

Falls der Schnatz noch nicht auf dem Spielfeld ist

```
"snitch": {
  "xPos": null,
```

```
  "yPos": null
}
```

9 Next

9.1 Description

Es wird an alle Clients eine Benachrichtigung für den nächsten Zug geschickt. Der Zug habende Spieler kann über die entityID herausgefunden werden. Nach einer Pause muss der Spieler wissen, wie viel Zeit er hat, um seinen Zug zu tätigen; dafür existiert das Feld `timeout`. Antwortet ein Client nicht innerhalb des `timeouts`, wird dies so aufgefasst, dass er keinen Zug macht und seine Spielfigur stehen bleibt, bzw. keine Aktion ausführt. Man kann somit das ganze Spiel über idle sein, ohne gekickt zu werden. Die Antwort des Clients muss innerhalb des `timeouts` am Server eintreffen um als Spielzug gewertet zu werden, ansonsten wird diese Nachricht ignoriert.

9.2 Type Declaration

9.2.1 `(string/entityID)`

- `leftSeeker`
- `leftKeeper`
- `leftChaser1`
- `leftChaser2`
- `leftChaser3`
- `leftBeater1`
- `leftBeater2`
- `rightSeeker`
- `rightKeeper`
- `rightChaser1`
- `rightChaser2`
- `rightChaser3`
- `rightBeater1`
- `rightBeater2`
- `snitch`
- `bludger1`
- `bludger2`
- `quaffle`
- `leftGoblin`
- `leftTroll`
- `leftElf`
- `leftNiffler`
- `leftWombat`
- `rightGoblin`
- `rightTroll`
- `rightElf`
- `rightNiffler`
- `rightWombat`

9.2.2 `(string/turnType)`

- `move`
- `action`
- `fan`
- `removeBan`

9.3 Definition

9.3.1 next

```
{
  "turn": "(string/entityID)",
  "type": "(string/turnType)",
  "timeout":"(int/millisec)"
}
```

9.4 Example

Spieler auf der linken Seite wird aufgefordert seinen 3. Jäger zu bewegen, innerhalb der nächsten 7.5 Sekunden

```
{
  "turn": "leftChaser3",
  "type": "move",
  "timeout":7500
}
```

10 Delta

10.1 Description

- Deltas werden geschickt um einen Zug zu machen (deltaRequest)
- Der Server kickt den Client bei falsch eintreffenden `deltaRequest`
- deltaBroadcast befinden sich nur im snapshot
- `null`-Felder müssen mitgeschickt werden. (Falls sie hier in der Defintion nicht angegeben sind werden sie immer auf null gesetzt)
- Bei verspätet eintreffenden Deltas werden diese ignoriert und der Server kickt den Client nicht!
- Die Deltas sollten jeden Zustandsuebergang abdecken. Es sollte also fuer den Client nach Verwertung des initialen Snapshots auch ohne Verwendung der folgenden Snapshots moeglich sein, den aktuellen Zustand korrekt zu erfassen.

deltaType	deltaRequest	deltaBroadcast
snitchCatch		x
bludgerBeating	x	x
quaffleThrow	x	x
snitchSnatch	x	x
trollRoar	x	x

deltaType	deltaRequest	deltaBroadcast
elfTeleportation	x	x
goblinShock	x	x
ban		x
bludgerKnockout		x
move	x	x
wrestQuaffle	x	x
foolAway		x
phaseChange		x
goalPointsChange		x
roundChange		x
skip	x	x
unban	x	x
turnUsed		x
wombatPoo	x	x
removePoo		x

10.2 Type Declaration

10.2.1 `(int/posx)`

- [0..16]

10.2.2 `(int/posy)`

- [0..12]

10.2.3 `(string/deltaType)`

- `snitchCatch`
- `bludgerBeating`
- `quaffleThrow`
- `snitchSnatch`
- `trollRoar`
- `elfTeleportation`
- `goblinShock`
- `ban`
- `bludgerKnockout`
- `move`
- `wrestQuaffle`
- `foolAway`
- `phaseChange`
- `goalPointsChange`
- `roundChange`
- `skip`
- `unban`
- `turnUsed`
- `wombatPoo`

- `removePoo`

10.2.4 `(string/entityID)`

- `leftSeeker`
- `leftKeeper`
- `leftChaser1`
- `leftChaser2`
- `leftChaser3`
- `leftBeater1`
- `leftBeater2`
- `rightSeeker`
- `rightKeeper`
- `rightChaser1`
- `rightChaser2`
- `rightChaser3`
- `rightBeater1`
- `rightBeater2`
- `snitch`
- `bludger1`
- `bludger2`
- `quaffle`
- `leftGoblin`
- `leftTroll`
- `leftElf`
- `leftNiffler`
- `leftWombat`
- `rightGoblin`
- `rightTroll`
- `rightElf`
- `rightNiffler`
- `rightWombat`

10.2.5 `(string/banReason)`

- `stooging`
- `blatching`
- `flacking`
- `haversacking`
- `snitchnip`
- `snitchSnatch`
- `elfTeleportation`
- `goblinShock`
- `trollRoar`
- `wombatPoo`

10.3 General Definition

- Der Client setzt das `success` Feld immer auf `null`. (deltaRequest)

- Der Server setzt das `success` Feld auf `true` falls es dem Wunsch des Clients entspricht und `false` wenn nicht. (deltaBroadcast)
- Der Server setzt das `success` Feld auf `null` falls es sich um eine Serverentscheidung handelt. (deltaBroadcast)

10.3.1 deltaRequest, deltaBroadcast

```
{
  "deltaType": "(string/deltaType)",
  "success": "(boolean)",
  "xPosOld":"(int/posx)",
  "yPosOld":"(int/posy)",
  "xPosNew":"(int/posx)",
  "yPosNew":"(int/posy)",
  "activeEntity": "(string/entityID)",
  "passiveEntity": "(string/entityID)",
  "phase": "(string/phaseType)",
  "leftPoints": "(int)",
  "rightPoints": "(int)",
  "round": "(int)",
  "banReason": "(string/banReason)"
}
```

10.4 DeltaType-Specific Definition

Felder die hier in der Defintion nicht angegeben sind werden immer auf null gesetzt und mitgeschickt. (siehe Beispiel)

10.4.1 `snitchCatch` (deltaBroadcast)

Wird nur vom Server gesendet(im Snapshot), da durch eine Bewegung vom Sucher auf das Feld des Schnatzes automatisch einen Versuch, den Schnatz zu fangen, ausgelöst wird. So kann auch kein "böser" Client das Spiel auf diese Art verändern. Diese Nachricht muss gesendet werden, wenn ein Sucher auf das Feld mit dem Schnatz zieht. Der Boolean "success" gibt entsprechend an, ob der Schnatz tatsächlich gefangen wurde.

10.4.1.1 case 1

```
# snitchCatch start (deltaBroadcast)
broadcast: next (left or right seeker should move)
request  : move (left or right seeker wants to move)
broadcast: snapshot (left or right seeker moves on field with snitch)
broadcast: snapshot (successful or unsuccessful catch)
# snitchCatch end (deltaBroadcast)
```

10.4.1.2 case 2

```
# snitchCatch start (deltaBroadcast)
broadcast: snapshot (snitch moves on field with left or right seeker)
```

```
broadcast: snapshot (successful or unsuccessful catch)
# snitchCatch end (deltaBroadcast)
```

10.4.1.3 Definition

```
{
  "deltaType": "snitchCatch",
  "success": "(boolean)", //If the catch was successful
  "activeEntity": "(string/entityID)", //Seeker the snitch collides with
  "leftPoints": "(int)",
  "rightPoints": "(int)"
}
```

10.4.2 `bludgerBeating` (deltaRequest)

```
# bludgerBeating start (deltaRequest)
broadcast: next (left or right beater should do action)
request  : deltaRequest (left or right beater wants a bludgerBeating)
# bludgerBeating end (deltaRequest)
```

```
{
  "deltaType": "bludgerBeating",
  "xPosNew": "(int/posx)", //New position of bludger
  "yPosNew": "(int/posy)",
  "activeEntity": "(string/entityID)", //Beater that beats the bludger
  "passiveEntity": "(string/entityID)" //Bludger that gets beaten
}
```

10.4.3 `bludgerBeating` (deltaBroadcast)

```
# bludgerBeating start (deltaBroadcast)
request  : deltaRequest (left or right beater wants a bludgerBeating)
broadcast: snapshot (left or right beater does a bludgerBeating)
# bludgerBeating end (deltaBroadcast)
```

```
{
  "deltaType": "bludgerBeating",
  "xPosOld": "(int/posx)", //Old position of bludger
  "yPosOld": "(int/posy)",
  "xPosNew": "(int/posx)", //New position of bludger
  "yPosNew": "(int/posy)",
  "activeEntity": "(string/entityID)", //Beater that beats the bludger
  "passiveEntity": "(string/entityID)" //Bludger that gets beaten
}
```

10.4.4 `quaffleThrow` (deltaRequest)

```
# quaffleThrow start (deltaRequest)
broadcast: next (chaser or keeper should do action)
request  : deltaRequest (chaser or keeper wants a quaffleThrow)
# quaffleThrow end (deltaRequest)
```

```
{
  "deltaType": "quaffleThrow",
  "xPosNew": "(int/posx)", //New position of the quaffle
  "yPosNew": "(int/posy)",
  "activeEntity": "(string/entityID)" //Player that throws the quaffle
}
```

10.4.5 `quaffleThrow` (deltaBroadcast)

```
# quaffleThrow start (deltaBroadcast)
request  : deltaRequest (chaser or keeper wants a quaffleThrow)
broadcast: snapshot (chaser or keeperr does a quaffleThrow)
# quaffleThrow end (deltaBroadcast)
```

```
{
  "deltaType": "quaffleThrow",
  "success": "(boolean)",
  "xPosOld": "(int/posx)", //Old position of the quaffle
  "yPosOld": "(int/posy)",
  "xPosNew": "(int/posx)", //New position of the quaffle
  "yPosNew": "(int/posy)",
  "activeEntity": "(string/entityID)", //Player that throws the quaffle
  "passiveEntity": "(string/entityID)" /*Player that catches the quaffle,
  null if noone caught it*/
}
```

10.4.6 `snitchSnatch` (deltaRequest)

```
# snitchSnatch start (deltaRequest)
broadcast: next (left or right niffler should act)
request  : deltaRequest (left or right niffler wants a snitchSnatch)
# snitchSnatch end (deltaRequest)
```

```
{
  "deltaType": "snitchSnatch",
}
```

10.4.7 `snitchSnatch` (deltaBroadcast)

```
# snitchSnatch start (deltaBroadcast)
request  : deltaRequest (left or right niffler wants a snitchSnatch)
broadcast: snapshot (left or right niffler does a snitchSnatch)
# snitchSnatch end (deltaBroadcast)
```

```
{
  "deltaType": "snitchSnatch",
  "xPosOld": "(int/posx)", //Old position of the snitch
  "yPosOld": "(int/posy)",
  "xPosNew": "(int/posx)", //New position of the snitch
  "yPosNew": "(int/posy)",
  "activeEntity": "(string/entityID)", //Niffler that snatches after the snitch
```

```
  "passiveEntity": "snitch"
}
```

10.4.8 `trollRoar` (deltaRequest)

```
# trollRoar start (deltaRequest)
broadcast: next (left or right troll should act)
request  : deltaRequest (left or right troll wants a trollRoar)
# trollRoar end (deltaRequest)
```

```
{
  "deltaType": "trollRoar",
}
```

10.4.9 `trollRoar` (deltaBroadcast)

```
# trollRoar start (deltaBroadcast)
request  : deltaRequest (left or right troll wants a trollRoar)
broadcast: snapshot (left or right troll does a trollRoar)
# trollRoar end (deltaBroadcast)
```

```
{
  "deltaType": "trollRoar",
  "xPosOld": "(int/posx)", //Old position of the quaffle
  "yPosOld": "(int/posy)",
  "xPosNew": "(int/posx)", //New position of the quaffle
  "yPosNew": "(int/posy)",
  "activeEntity": "(string/entityID)", //Troll that roar
  "passiveEntity":  "(string/entityID)" /* entityID that holds the quaffle,
   null if quaffle is not held */
}
```

10.4.10 `elfTeleportation` (deltaRequest)

```
# elfTeleportation start (deltaRequest)
broadcast: next (left or right elf should act)
request  : deltaRequest (left or right elf wants a elfTeleportation)
# elfTeleportation end (deltaRequest)
```

```
{
  "deltaType": "elfTeleportation",
  "passiveEntity": "(string/entityID)" //Entity that gets teleported by the elf
}
```

10.4.11 `elfTeleportation` (deltaBroadcast)

Falls ein Spieler einen Quaffel hält wird dieser nicht mit teleportiert. Teleportieren betrifft nur den Spieler, seine Kleidung und seinen Besen.

Der Quaffel bleibt auf dem Feld liegen.

```
# elfTeleportation start (deltaBroadcast)
request  : deltaRequest (left or right elf wants a elfTeleportation)
broadcast: snapshot (left or right elf does a elfTeleportation)
# elfTeleportation end (deltaBroadcast)
```

```
{
  "deltaType": "elfTeleportation",
  "xPosOld": "(int/posx)", //Old position of the passive entity
  "yPosOld": "(int/posy)",
  "xPosNew": "(int/posx)", //New position of the passive entity
  "yPosNew": "(int/posy)",
  "activeEntity": "(string/entityID)", //Elf that does the teleportation
  "passiveEntity": "(string/entityID)" //Entity that gets teleported by the elf
}
```

10.4.12 goblinShock (deltaRequest)

```
# goblinShock start (deltaRequest)
broadcast: next (left or right goblin should act)
request  : deltaRequest (left or right goblin wants a goblinShock)
# goblinShock end (deltaRequest)
```

```
{
  "deltaType": "goblinShock",
  "passiveEntity": "(string/entityID)" //Passive entity that gets shocked
}
```

10.4.13 goblinShock (deltaBroadcast)

Falls die Entity den Quaffel hält vertändelt sie diesen zuerst mit einem foolAway dannach kommt der goblinShock.

10.4.13.1 case 1 without quaffle

```
# goblinShock start (deltaBroadcast)
request  : deltaRequest (left or right goblin wants a goblinShock)
broadcast: snapshot (left or right goblin does a goblinShock)
# goblinShock end (deltaBroadcast)
```

10.4.13.2 case 2 if the entity holds a quaffle

```
# goblinShock start (deltaBroadcast)
request  : deltaRequest (left or right goblin wants a goblinShock)
broadcast: snapshot (entity foolAway the quaffle 100% certain)
broadcast: snapshot (left or right goblin does a goblinShock)
# goblinShock end (deltaBroadcast)
```

```
{
  "deltaType": "goblinShock",
  "xPosOld": "(int/posx)", //Old position of the passive entity
  "yPosOld": "(int/posy)",
```

```
  "xPosNew": "(int/posx)", //New position of the passive entity
  "yPosNew": "(int/posy)",
  "activeEntity": "(string/entityID)", //Goblin that shocks the passive entity
  "passiveEntity": "(string/entityID)" //Passive entity that gets shocked
}
```

10.4.14 `ban` (deltaBroadcast)

Sobald ein Tor Fällt werden die gebannten Spieler, nach der Fanphase der entsprechenden Runde wieder auf das Spielfeld gelassen. Dabei fordert der Server jede Spielfigur einzeln auf sich auf das Spielfeld zu begeben. Falls keine Entscheidung getroffen wird entscheidet der Server und setzt die Spielfigur zufällig auf das Spielfeld.

- Ändert am Snapshot: setzt das `ban` Flag auf true

```
# ban start (deltaBroadcast)
broadcast: snapshot (entity which does something against the rules)
broadcast: snapshot (entity gets banned)
# ban end (deltaBroadcast)
```

```
{
  "deltaType": "ban",
  "passiveEntity": "(string/entityID)", //Entity that gets banned
  "banReason": "(string/banReason)"
}
```

10.4.15 `bludgerKnockout` (deltaBroadcast)

10.4.15.1 case 1 passive bludgerKnockout

beater sind immune gegenüber passiven bludgerKnockouts.

```
# bludgerKnockout start (deltaBroadcast)
broadcast: snapshot (in the ballPhase bludger moves to the field with the entity)
broadcast: snapshot (entity stands on the field and gets  bludgerKnockout)
# bludgerKnockout end (deltaBroadcast)
```

10.4.15.2 case 2 active bludgerKnockout

Beater können nur Seeker, Keeper und Chaser ausknocken.

```
# bludgerKnockout start (deltaBroadcast)
broadcast: snapshot (beater does a bludgerBeating)
broadcast: snapshot (entity stands on the field and gets  bludgerKnockout)
# bludgerKnockout end (deltaBroadcast)
```

10.4.15.3 case 3 passive bludgerKnockout with Quaffle

beater sind immune gegenüber passiven bludgerKnockouts.

```
# bludgerKnockout start (deltaBroadcast)
broadcast: snapshot (in the ballPhase bludger moves to the field with the entity)
broadcast: snapshot (foolAway)
```

```
broadcast: snapshot (entity stands on the field and gets  bludgerKnockout)
# bludgerKnockout end (deltaBroadcast)
```

10.4.15.4 case 4 active bludgerKnockout with Quaffle

Beater können andere Beater ausknocken.

```
# bludgerKnockout start (deltaBroadcast)
broadcast: snapshot (beater does a bludgerBeating)
broadcast: snapshot (foolAway)
broadcast: snapshot (entity stands on the field and gets  bludgerKnockout)
# bludgerKnockout end (deltaBroadcast)
```

```
{
  "deltaType": "bludgerKnockout",
  "success": "(boolean)",
  "xPosOld": "(int/posx)", /*Old position where the bludger possibly
  kocks out the passive entity*/
  "yPosOld": "(int/posy)",
  "xPosNew": "(int/posx)", /*New position of the bludger after a
  successful knockout. null if success false*/
  "yPosNew": "(int/posy)",
  "activeEntity": "(string/entityID)", /*The bludger that knocks
  the passive entity out */
  "passiveEntity": "(string/entityID)" //The passive entity that gets knocked out
}
```

10.4.16 `move` (deltaRequest)

Antwort auf ein `next` mit type move

```
# move start (deltaRequest)
broadcast: next (entity should move)
request  : deltaRequest (entity wants to move)
# move end (deltaRequest)
```

```
{
  "deltaType": "move",
  "xPosNew": "(int/posx)", //New position of the active entity
  "yPosNew": "(int/posy)",
  "activeEntity": "(string/entityID)" //Entity that gets a new position
}
```

10.4.17 `move` (deltaBroadcast)

- Ändert am Snapshot: Snitch und Bludger werden bewegt, Spieler werden bewegt. Falls Quaffel oder Bluger getragen werden werden sie auch bewegt

10.4.17.1 case 1 playerPhase

```
# move start (deltaBroadcast)
request  : deltaRequest (entity wants to move)
```

```
broadcast: snapshot (entity moves)
# move end (deltaBroadcast)
```

10.4.17.2 case 2 ballPhase

```
# move start (deltaBroadcast)
broadcast: snapshot (ball moves)
# move end (deltaBroadcast)
```

10.4.17.3 case 3 Quaffle on GoalField

```
# move start (deltaBroadcast)
broadcast: snapshot (Quaffle was moved to the GoalField)
broadcast: snapshot (Quaffle moves away to keeper or middle)
# move end (deltaBroadcast)
```

10.4.17.4 case 4 Quaffle on GoalField with Points

```
# move start (deltaBroadcast)
broadcast: snapshot (Quaffle was moved to the GoalField)
broadcast: snapshot (goalPointsChange)
broadcast: snapshot (Quaffle Quaffle moves away to keeper or middle)
# move end (deltaBroadcast)
```

10.4.17.5 case 5 blatching

```
# move start (deltaBroadcast)
broadcast: snapshot (entity moves and produce a blatching foul)
broadcast: snapshot (entity moves because of blatching)
# move end (deltaBroadcast)
```

```
{
  "deltaType": "move",
  "xPosOld": "(int/posx)", //Old position of the active entity
  "yPosOld": "(int/posy)",
  "xPosNew": "(int/posx)", //New position of the active entity
  "yPosNew": "(int/posy)",
  "activeEntity": "(string/entityID)" //Entity that gets a new position
}
```

10.4.18 `wrestQuaffle` (deltaRequest)

```
# wrestQuaffle start (deltaRequest)
broadcast: next (chaser or keeper should do action)
request  : deltaRequest (chaser or keeper wants a wrestQuaffle)
# wrestQuaffle end (deltaRequest)
```

```
{
  "deltaType": "wrestQuaffle",
  "activeEntity": "(string/entityID)" //Entity that wrest the Quaffle
}
```

10.4.19 `wrestQuaffle` (deltaBroadcast)

- Ändert am Snapshot: ändert die Position des Quaffels und setzt gegebenenfalls den Quaffelträger neu

```
# wrestQuaffle start (deltaBroadcast)
request  : deltaRequest (chaser or keeper wants a wrestQuaffle)
broadcast: snapshot (successful or unsuccessful wrestQuaffle)
# wrestQuaffle end (deltaBroadcast)
```

```
{
  "deltaType": "wrestQuaffle",
  "success": "(boolean)",
  "xPosOld": "(int/posx)", //Old position of the quaffle
  "yPosOld": "(int/posy)",
  "xPosNew": "(int/posx)", //New position of the quaffle
  "yPosNew": "(int/posx)",
  "activeEntity": "(string/entityID)", //Entity that wrest the Quaffle
  "passiveEntity": "(string/entityID)" //Entity that loses the Quaffle
}
```

10.4.20 `foolAway` (deltaBroadcast)

- Ändert am Snapshot: ändert die Position des Quaffels und setzt gegebenenfalls den Quaffelträger neu

foolAway wird nur in Verbindung mit anderen deltaBroadcast gesendet, wie d.h. goblinShock, bludgerKnockout und hat dort immer 100% Wahrscheinlichkeit.

```
{
  "deltaType": "foolAway",
  "xPosOld": "(int/posx)", //Old position of the quaffle
  "yPosOld": "(int/posy)",
  "xPosNew": "(int/posx)", //New position of the quaffle
  "yPosNew": "(int/posy)",
  "activeEntity": "quaffle",
  "passiveEntity": "(string/entityID)" //Entity that loses the Quaffle
}
```

10.4.21 `phaseChange` (deltaBroadcast)

- Ändert am Snapshot: kann die phase zu `unbanPhase`, `playerPhase`, `fanPhase`, `gameFinish` ändern. (Für ballPhase ist roundChange zuständig)

10.4.21.1 case 1 toPlayerPhase

```
# phaseChange start (deltaBroadcast)
broadcast: snapshot (Quaffle not on goal field, ballPhase und alle bälle wurden bewegt)
broadcast: snapshot (phaseChange to playerPhase)
# phaseChange end (deltaBroadcast)
```

10.4.21.2 case 2 toFanPhase

```
# phaseChange start (deltaBroadcast)
broadcast: snapshot (Quaffle not on goal field, playerPhase, alle Spieler
    (turnUsed == true) oder (ban == true) oder (knockout == true))
broadcast: snapshot (phaseChange to fanPhase)
# phaseChange end (deltaBroadcast)
```

10.4.21.3 case 3 toUnbanPhase

```
# phaseChange start (deltaBroadcast)
broadcast: snapshot (Quaffle not on goal field, fanPhase, every fan has done
    his (turnUsed==true) oder (ban == true))
broadcast: snapshot (phaseChange to unbanPhase)
# phaseChange end (deltaBroadcast)
```

10.4.21.4 case 4 toGameFinish

```
# phaseChange start (deltaBroadcast)
broadcast: snapshot (something happen which finish the game)
broadcast: snapshot (phaseChange to gameFinish)
# phaseChange end (deltaBroadcast)
```

```
{
  "deltaType": "phaseChange",
  "phase": "(string/phaseType)"
}
```

10.4.22 `goalPointsChange` (deltaBroadcast)

- Ändert am Snapshot: setzt die Points der jeweiligen Teams, set das `goalWasThrownThisRound` Flag auf true

```
{
  "deltaType": "goalPointsChange",
  "leftPoints": "(int)",
  "rightPoints": "(int)",
}
```

10.4.23 `roundChange` (deltaBroadcast)

- Ändert am Snapshot: ändert die phase zu `ballPhase`, entbannt falls ein Tor geschossen (`goalWasThrownThisRound`) wurde alle Spieler, setzt alle `turnUsed` Flags auf false, zählt die `round` hoch. Setzt das `goalWasThrownThisRound` auf false

```
# roundChange start (deltaBroadcast)
broadcast: snapshot (unbanPhase, alle Spieler die keinen ban haben sind auf
    dem Spielfeld)
broadcast: snapshot (roundChange to ballPhase)
# roundChange end (deltaBroadcast)
```

```
{
  "deltaType": "roundChange",
  "round": "(int)"
}
```

10.4.24 `skip` (deltaRequest,deltaBroadcast)

Beendet einen durch einen next angeforterten Zug vorzeitig oder Client hat nicht in der timout-Zeit geantwort und Server beendet damit den Zug.

- Ändert am Snapshot: nichts

10.4.24.1 case 1 active skip from client

```
# skip start (deltaRequest,deltaBroadcast)
broadcast: next (entity should do something)
request  : deltaRequest (skip)
broadcast: snapshot (skip)
# skip end (deltaRequest,deltaBroadcast)
```

10.4.24.2 case 2 no answer from client after timeout

```
# skip start (deltaBroadcast)
broadcast: next (entity should do something)
broadcast: snapshot (skip)
# skip end (deltaBroadcast)
```

```
{
  "deltaType": "skip",
  "activeEntity": "(string/entityID)" //entity that gets skipped
}
```

10.4.25 `unban` (deltaRequest, deltaBroadcast)

Setzt die neue position eines Spielers nach einem Next removeBan

- Ändert am Snapshot: setzt die Position des Spielers und ban = false

(Ablauf analog zum move)

```
{
  "deltaType": "unban",
  "xPosNew": "(int/posx)", //New position of the player
  "yPosNew": "(int/posx)",
  "activeEntity": "(string/entityID)" //Entity that gets unbanned
}
```

10.4.26 `turnUsed` (deltaBroadcast)

Damit angezeigt werden kann das ein Spieler nach seinen ein oder zwei Zügen und der Action verbraucht ist.

- Ändert am Snapshot: das `turnUsed` Flag des Fans oder Spielers wird auf true gesetzt.

Es sollte immer nur eine Entity am Zug sein. Der Zug beschreibt den Vorgang vom ersten next bist zum turnUsed. Es alle möglichen actionen müssen vom Server angeboten werden. Erst wenn es keine weiteren Möglichkeiten mehr gibt (move, action, fan) gibt, kann das turnUsed vom Server geschickt werden.

10.4.26.1 case 1 single move

```
# turnUsed start (deltaBroadcast)
broadcast: next (entity should move)
broadcast: snapshot (entity move)
...
broadcast: snapshot (entity turnUsed)
# turnUsed end (deltaBroadcast)
```

10.4.26.2 case 2 double move

```
# turnUsed start (deltaBroadcast)
broadcast: next (entity should move)
broadcast: snapshot (entity move)
...
broadcast: snapshot (entity move)
...
broadcast: snapshot (entity turnUsed)
# turnUsed end (deltaBroadcast)
```

10.4.26.3 case 3 single move and action

```
# turnUsed start (deltaBroadcast)
broadcast: next (entity should move)
broadcast: snapshot (entity move)
...
broadcast: snapshot (entity action)
...
broadcast: snapshot (entity turnUsed)
# turnUsed end (deltaBroadcast)
```

10.4.26.4 case 4 double move and action

```
# turnUsed start (deltaBroadcast)
broadcast: next (entity should move)
broadcast: snapshot (entity move)
...
broadcast: snapshot (entity move)
...
broadcast: snapshot (entity action)
...
broadcast: snapshot (entity turnUsed)
# turnUsed end (deltaBroadcast)
```

10.4.26.5 case 5 fan

```
# turnUsed start (deltaBroadcast)
broadcast: next (fan should do fan stuff)
broadcast: snapshot (fan do fan stuff)
broadcast: snapshot (entity turnUsed)
# turnUsed end (deltaBroadcast)
```

```
{
  "deltaType": "turnUsed",
  "activeEntity": "(string/entityID)"
}
```

10.4.27 `wombatPoo` (deltaRequest)

- Kann in der Fanphase nach einem next geschickt werden

```
# wombatPoo start (deltaRequest)
broadcast: next (left or right wombat should act)
request  : deltaRequest (left or right wombat wants a wombatPoo)
# wombatPoo end (deltaRequest)
```

```
{
  "deltaType": "wombatPoo",
  "xPosNew":"(int/posx)", // position of the cube
  "yPosNew":"(int/posy)"
}
```

10.4.28 `wombatPoo` (deltaBroadcast)

- Kann in der Fanphase geschickt werden
- Ändert am Snapshot: Es wird ein Würfel in die Liste `wombatCubes` hinzugefügt

```
# wombatPoo start (deltaBroadcast)
request  : deltaRequest (left or right wombat wants a wombatPoo)
broadcast: snapshot (left or right wombat does a wombatPoo)
# wombatPoo end (deltaBroadcast)
```

```
{
  "deltaType": "wombatPoo",
  "xPosNew":"(int/posx)", // position of the cube
  "yPosNew":"(int/posy)",
  "activeEntity": "(string/entityID)" // wombat
}
```

10.4.29 `removePoo` (deltaBroadcast)

- Kann jederzeit geschickt werden
- Ändert am Snapshot: Es wird ein Würfel oder alle Wuerfel aus der Liste `wombatCubes` entfernt

```
{
  "deltaType": "removePoo",
  "xPosOld":"(int/posx)", // position of the cube(s) to be removed
  "yPosOld":"(int/posy)"     //if posOld is null ALL cubes are removed
  "activeEntity": "(string/entityID)" /*entity that landed on the field position
  of a cube and triggered the removal */
}
```

Es gibt also 2 Faelle:

1) Wird gesendet wenn eine neue Fanphase *beginnt* (um komplizierte Serverlogik einzusparen)

```
{
  "deltaType": "removePoo",
  "xPosOld": null
  "yPosOld": null
  "activeEntity": null
}
```

2) Hier wurde ein Spieler von einem Elf auf ein Feld mit Poo teleportiert, wodurch die Poo entfernt wird. (die ausloesende Aktion ist separat zu schicken und ausgelassene Felder sind null)

```
{
  "deltaType": "removePoo",
  "xPosOld": 4
  "yPosOld": 5
  "activeEntity": leftSeeker
}
```

10.5 Example

10.5.1 Example `quaffleThrow`

User sendet seine Entscheidung

```
{
  "deltaType": "quaffleThrow",
  "success": null,
  "xPosOld":null,
  "yPosOld":null,
  "xPosNew":6,
  "yPosNew":6,
  "activeEntity": "leftChaser1",
  "passiveEntity": null,
  "phase": null,
  "leftPoints": null,
  "rightPoints": null,
  "round": null,
  "banReason": null
}
```

broadcast: Durch Jäger des linken Spielers auf Feld 4,4 werfen, nicht erfolgreich, abgefangen durch Spieler auf Position 5,5. Durch die passiveEntity ist klar schließbar wer den Ball abgefangen

hat. Wenn die passiveEntity kein Chaser oder Keeper ist wird danach ein foolAway geschickt.


```
{
  "deltaType": "quaffleThrow",
  "success": false,
  "xPosOld": 4,
  "yPosOld": 4,
  "xPosNew": 5,
  "yPosNew": 5,
  "activeEntity": "leftChaser1",
  "passiveEntity": "rightChaser3",
  "phase": null,
  "leftPoints": null,
  "rightPoints": null,
  "round": null,
  "banReason": null
}
```


10.5.2 Example `ban`

Server verbannt Spieler leftChaser1 vom Spielfeld


```
{
  "deltaType": "ban",
  "success": null,
  "xPosOld": null,
  "yPosOld": null,
  "xPosNew": null,
  "yPosNew": null,
  "activeEntity": null,
  "passiveEntity": "leftChaser1",
  "phase": null,
  "leftPoints": null,
  "rightPoints": null,
  "round": null,
  "banReason": null
}
```


Der Server fordert auf einen Zug zu machen


```
{
  "turn": "leftChaser1",
  "type": "removeBan",
  "timeout":7500
}
```


Der Client sendet seine Entscheidung


```
{
  "deltaType": "unban",
  "success": null,
  "xPosOld": null,
  "yPosOld": null,
  "xPosNew": 5,
```

```
  "yPosNew": 5,
  "activeEntity": "leftChaser1",
  "passiveEntity": null,
  "phase": null,
  "leftPoints": null,
  "rightPoints": null,
  "round": null,
  "banReason": null
}
```


Der Server antwortet.


```
{
  "deltaType": "unban",
  "success": null,
  "xPosOld": null,
  "yPosOld": null,
  "xPosNew": 5,
  "yPosNew": 5,
  "activeEntity": "leftChaser1",
  "passiveEntity": null,
  "phase": null,
  "leftPoints": null,
  "rightPoints": null,
  "round": null,
  "banReason": null
}
```


11 Pause

11.1 Description

Eine Pause-Request kann von allen spielenden Clients an den Server gesendet werden. Außerdem können alle spielenden Clients eine Continue-Request an den Server senden.

Nach jeder Request, broadcastet der Server eine Pause-Response an alle Clients(auch Zuschauer), die einen boolschen Wert enthält, der den aktuellen Pause-Zustand (Spiel pausiert: pause = true oder Spiel nicht pausiert: pause = false) enthält. Der Server regelt also die komplette Logik und ändert den Pause-Zustand anhand der eingehenden Requests wie im Lastenheft vorgegeben.

Das Lastenheft definiert folgende Serverlogik: “Falls ein mitspielender Client eine Pausierung der Partie wünscht, unterbricht der Server diese, bis einer der Mitspieler anzeigt, dass er weiterspielen möchte. KI-Clients dürfen keine Pausen verlangen.”

Der Server überprüft also bei einer Pause-Request ob die empfangene userName mit einem spielenden Client übereinstimmt, wenn ja pausiert er das Spiel, falls es nicht schon pausiert ist. Analog überprüft er die userName wenn er eine Continue-Request empfängt, wenn diese mit einem spielenden Client übereinstimmt, beendet er die Pause, falls das Spiel nicht schon läuft. In allen Fällen broadcastet der Server eine Pause-Response an alle Clients.

11.2 Definition

Annmerkung: das message feld kann frei ausgefüllt werden. z.B. Ich bin gleich wieder da, muss nur mal kurz die Welt retten.

11.2.1 pauseRequest

```
{
  "message" : "(string)"
}
```

11.2.2 continueRequest

```
{
  "message" : "(string)"
}
```

11.2.3 pauseResponse

```
{
  "message": "(string)",
  "userName":"(string/userNameType)",
  "pause": "(boolean)"
}
```

12 Reconnect

12.1 Description

Wird mit unicast an Client gesendet, wenn er sich bei laufendem Spiel einloggt. d.h. Nachdem `matchStart` versendet wurde. So erhält der Zuschauer oder Spieler welcher später sich zu dem Spiel verbindet mit nur einem json die gesamten Informationen.

12.2 Definition

12.2.1 reconnect

```
{
  "matchStart": {
    "timestamp": "(string/yyyy-MM-dd HH:mm:ss.SSS)", //time of the first Snapshot
    "payloadType": "matchStart",
    "payload": "(json/matchStart)"
  },
  "snapshot": {
    "timestamp": "(string/yyyy-MM-dd HH:mm:ss.SSS)",
    "payloadType": "snapshot",
    "payload": "(json/snapshot)"
```

```
  },
  "next": {
    "timestamp": "(string/yyyy-MM-dd HH:mm:ss.SSS)",
    "payloadType": "next",
    "payload": "(json/next)"
  }
}
```

Falls es kein next gibt wird dies so auf null gesetzt

```
"next": null
```

12.2.2 Example

```
# Reconnect start
request  : joinRequest
unicast  : joinResponse
broadcast: loginGreeting
unicast  : reconnect
# Reconnect end
```

13 Replay

13.1 Description

Das Replay besteht aus statischen Informationen wie matchConfig,leftTeamConfig, alle Spieler welche mindesten einmal das Spiel betreten haben und einer log-Liste von Deltas. Der Client kann das Replay nur bekommen wenn er sich noch mit der Lobby befindet in welcher das Spiel statt findet. Das Replay wird in einem file persistent abgespeichert. Das Replay ist erst nach dem MatchFinish anforderbar. Das getReplay bezieht sich immer auf das letzte MatchFinish, falls es noch kein Spiel in dieser lobby gab wird der Client gekickt.

13.2 Type Declaration

13.2.1 `(string/replayPayloadType)` (subset of payloadType)

- `deltaBroadcast`
- `matchFinish` (letztes Objekt in dem log array)

13.3 Definition

13.3.1 getReplay

```
{

}
```

13.3.2 replay (file)

```
{
  "lobby": "(string/lobbyType)",
  "startTimestamp":"(string/yyyy-MM-dd HH:mm:ss.SSS)",
  "matchConfig":"(json/matchConfig)",
  "leftTeamConfig":"(json/teamConfig)",
  "rightTeamConfig":"(json/teamConfig)",
  "leftTeamUserName":"(string/userNameType)",
  "rightTeamUserName":"(string/userNameType)",
  "spectatorUserName": [
    "(string/userNameType)"
  ],
  "firstSnapshot":"(json/snapshot)",
  "log": [
    {
      "timestamp": "(string/yyyy-MM-dd HH:mm:ss.SSS)",
      "payloadType": "(string/replayPayloadType)",
      "payload": "(json)"
    }
  ]
}
```

13.4 Error and Warning {#Error and Warning}

13.4.1 Description

- Error: Bei einem Fehler welcher es erfordert das der Server die Verbindung zum Client abbricht. Der Server sendet über unicast. Das Feld `violationOfProtocol` zeigt ob es ich um eine Verletzung der Spielregen handelt. (Error entspricht Verbindungsabbruch.)
- Warning: Bei einem Fehler welcher vom Server bemerkt wird, der Client wird informiert, weiter passiert nichts. Der Server sendet ein warning (Bei Warning bleit Verbindung erhalten.)

13.4.2 Definition

13.4.3 privateError

```
{
  "code":"(int)",
  "information" : "(string)",
  "triggerMatchFinish" : "(boolean)"
}
```

13.4.4 privateWarning

```
{
  "code":"(int)",
  "information" : "(string)"
}
```

13.4.5 Error and Warning List

Hier sind einige Fälle definiert welche eintretten können. Es kann sein das zwei Fälle auftretten könnnen. Dann kann frei gewählt werden. Es gilt allgemein je besser die Meldung desto eher kann man herrausfinden wo das Problem liegt.

Man muss nicht alle Codes implementieren. Falls man Codes für das eigene Team schreiben möchte bitte bei z.B. für Team 5 bei 500 anfangen und per merge-request dem Komitee mitteilen. (alles bitte in Englisch schreiben.)

Die Codes sind dreistellig:

ABC

where A is either of one:

1 for error that triggered MatchFinish,

2 for errors that didn't trigger MatchFinish (replace (boolean) with true or false depending on whether error was produced by player or other user)

3 for warnings.

Note that the indices 150-199 and 250-299 are reserved for those cases that differ only in their boolean value, so every 151 is equal to 251 regarding the string and error type, just the boolean value is different. This allows the server to be able to e.g. add 100 to make a non-match-finishing error from a possibly match-finishing error.

B and C are numbers from 0..99 and simply incremented indices.

```
{
  "code":200, //or 100
  "information" : "Error: server has some reason to kick you (but case is undefined)",
  "triggerMatchFinish" : (boolean)
}

{
  "code": 300
  "information" : "some definable string for undefined case",
}

{
  "code":101,
  "information" : "Error:  invalid teamFormation by a player",
  "triggerMatchFinish" : true
}

{
  "code":102,
  "information" : "Error: invalid deltaRequest by active player"
  "triggerMatchFinish" : true
}

{
  "code":201,
  "information" : "Error: joinRequest-doublelogin: User logs in from two clients
   at the same time. The last login is always executed. Error
  goes to the old logon, Warning goes to the new logon.",
```

```
   "triggerMatchFinish" : false
}

{
  "code":202,
  "information" : "Error:  incompatibleUserMods, the server does not supports the
  requested mods",
  "triggerMatchFinish" : false
}

{
  "code":203,
  "information" : "Error:  invalid username",
  "triggerMatchFinish" : false
}

{
  "code":204,
  "information" : "Error:  invalid lobbyname",
  "triggerMatchFinish" : false
}

{
  "code":205,
  "information" : "Error:  invalid password (too long or forbidden characters)",
  "triggerMatchFinish" : false
}

{
  "code":206,
  "information" : "Error:  wrong password",
  "triggerMatchFinish" : false
}

{
  "code":207,
  "information" : "Error:  invalid teamConfig by player",
  "triggerMatchFinish" : false
}

{
  "code":208,
  "information" : "Error: user sent non-joinRequest-message before joinResponse",
  "triggerMatchFinish" : false
}

{
  "code": 151, //and 251
  "information" : "Error:  Server could not parse json",
  "triggerMatchFinish" : (boolean)
}

{
  "code": 152, //and 252
  "information" : "Error:  Server could parse a json but it is not in the
   Container format.",
```

```
  "triggerMatchFinish" : (boolean)
}

{
  "code": 153, //and 253
  "information" : "Error:  The container format could be recognized,
  but the payload does not correspond to the desired format (e.g. missing
  Fields or fields have wrong types e.g. String instead of Int)",
  "triggerMatchFinish" : (boolean)
}

{
  "code": 154, //and 254
  "information" : "Error:  payloadType not allowed at this point
  (e.g. teamFormation or deltaRequest from spectator)"
  "triggerMatchFinish" : (boolean) /*false for spectators, but if a player
  sends a second teamFormation this will be true */
}

{
  "code":301,
  "information" : "Warning: joinRequest-doublelogin: User logs in from two clients
  at the same time. The last login is always executed. Error
  goes to the old logon, Warning goes to the new logon."
}

{
  "code":302,
  "information" : "Warning:  teamConfig sent while game is running"
}

{
  "code":303,
  "information" : "Warning: It's not your turn - deltaRequest on other
   player's turn."
}

{
  "code":304,
  "information" : "Warning: deltaRequest received while game is paused, please
   make a break"
}
```

14 standardisierte Mods (modifications)

14.1 Description

Es ist nicht verpflichtend Mods zu implementieren. Falls der Server und der userClient die Mods unterstützen können diese im login so aktiviert werden

z.B. `{"mods":["chat","replayWithSnapshot"]}`

14.1.1 Overview

request	broadcast	unicast	siehe
sendChat	globalChat	reconnectChat	siehe Chat-mod
			siehe disableGenderBalance-mod
getReplayWithSnapshot		replayWithSnapshot	siehe ReplayWithSnapshot-mod
		lobbies	siehe Lobby-mod
testRequest		testResponse	siehe Test-mod

14.1.2 Liste von Artefakten (werden im Container-Format persistent abgespeichert)

- testFormat
- replayWithSnapshot

14.2 Chat-mod

14.2.1 Description

Damit man sich mit anderen unterhalten kann. Zuschauer und Spielende können in den Chat schreiben. der Server broadcastet dann an alle gerade verbundenen Spieler. Wird im login so aktiviert `{"mods":["chat"]}`

14.2.2 Definition

14.2.2.1 sendChat

```
{
  "information" : "(string)"
}
```

14.2.2.2 globalChat

```
{
  "senderUserName": "(string/userNameType)",
  "information" : "(string)"
}
```

14.2.2.3 reconnectChat

Sendet die letzten max 10 Nachrichten nach einem Reconnect. Die neuste Chat Nachricht ist am Ende das Arrays.

```
{
  "lastTenChat" : [
    {
    "senderUserName": "(string/userNameType)",
    "information" : "(string)"
    }
  ]
}
```

14.3 DisableGenderBalance-mod

14.3.1 Description

Schaltet die Regel das Teams nur maximal eine bestimmte Anzahl von männlichen und weiblichen Spielfiguren haben dürfen aus. Wird im login so aktiviert `{"mods":["disableGenderBalance"]}`

14.4 ReplayWithSnapshot-mod

14.4.1 Description

Wie Replay nur alle snapshot und next sind auch enthalten. Wird im login so aktiviert `{"mods":["replayWithSnapshot"]}`

14.4.2 Type Declaration

14.4.2.1 `(string/replayPayloadType2)` (subset of payloadType)

- `matchStart`
- `snapshot`
- `next`
- `matchFinish` (letztes Objekt in dem log array)

14.4.3 Definition

14.4.3.1 getReplayWithSnapshot

```
{

}
```

14.4.3.2 replayWithSnapshot (file)

```
{
  "lobby": "(string/lobbyType)",
  "startTimestamp":"(string/yyyy-MM-dd HH:mm:ss.SSS)",
  "matchConfig":"(json/matchConfig)",
  "leftTeamConfig":"(json/teamConfig)",
  "rightTeamConfig":"(json/teamConfig)",
  "leftTeamUserName":"(string/userNameType)",
  "rightTeamUserName":"(string/userNameType)",
  "spectatorUserName": [
    "(string/userNameType)"
  ],
  "firstSnapshot":"(json/snapshot)",
  "log": [
    {
      "timestamp": "(string/yyyy-MM-dd HH:mm:ss.SSS)",
      "payloadType": "(string/replayPayloadType2)",
      "payload": "(json)"
```

```
    }
  ]
}
```

14.5 Lobby-mod

Wird so aktiviert `{"mods":["lobbies"]}`

Der Server sendet *on Connect* d.h. vor der JoinRequest folgende lobbyList

14.5.1 lobbies

```
{
  "lobbies" : [
    {
      "name": "(string/lobbyName)",
      "matchStarted": boolean,
      "connectedUsers": int
    }
  ],
}
```

14.6 Test-mod

Wird im login so aktiviert `{"mods":["test"]}`

14.6.1 testFormat

```
{
 "testType":"snapshot-delta-snapshot", // noch keine Anderen Typen
 "createdByTeamXX":"(string)", // z.b. team99
 "description":"(string)", // z.b. Mein erster Test
 "snapshot" : "(json/snapshot)",
 "delta" : "(json/deltaBroadcast)",
 "expectedSnapshot" : "(json/snapshot)"
}
```

14.6.2 testRequest

```
{
 "testType":"snapshot-delta-snapshot", // noch keine Anderen Typen
 "createdByTeamXX":"(string)", // z.b. team99
 "description":"(string)", // z.b. Mein erster Test
 "snapshot" : "(json/snapshot)",
 "delta" : "(json/deltaBroadcast)",

}
```

14.6.3 testResponse

```
{
  "actualSnapshot" : "(json/snapshot)"
}
```

15 appendix

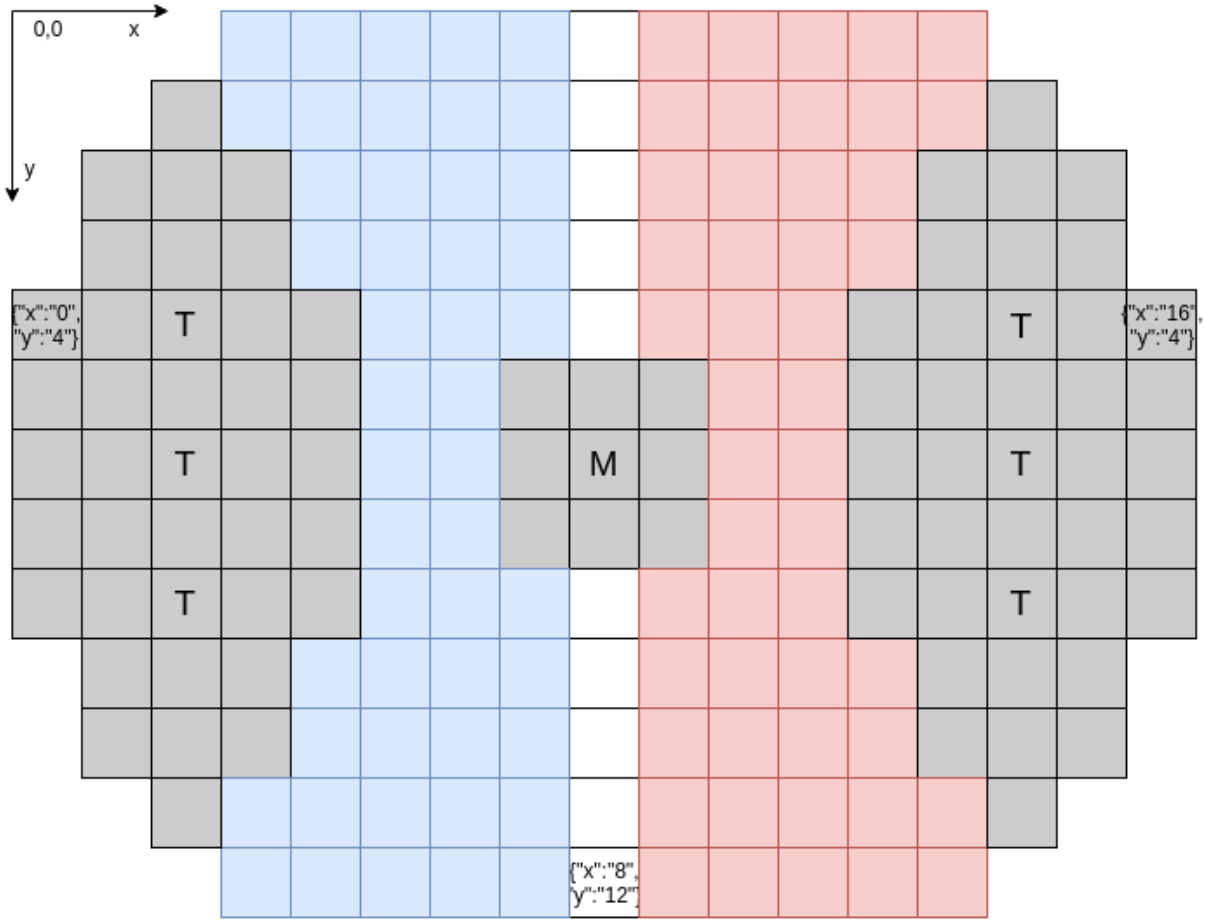

Figure 1: QuidditchPitch Positionen. x und y definiert. Blau und Rot plus Hüterzone entspricht den möglichen Startpositionen der Spielfiguren. Torfelder sind keine gültingen Startpositionen.