Spieleserver

Der Spieleserver ist die zentrale Komponente von Dynamite Boy und kompatiblen Spielen, da dieser die komplette Organisation und Operation der Spielesitzungen übernimmt.

Konkret ist die Operation des Spieleservers in zwei Teile eingeteilt:

- Operation einer Lobby (im Weiteren als "Lobbymodus" beschrieben), zu der Spieler beitreten können, von denen einer zum Spielleiter erklärt wird, der die nächste Spielerunde konfigurieren und starten darf
- Operation einer Spielerunde (im Weiteren als "Spielemodus" beschrieben), wobei das konkrete Spiel komplett auf dem Spieleserver ausgeführt wird und Clients nur für Anzeige und Steuerung zuständig sind

Verbindungen zwischen dem Spieleserver und den teilnehmenden Spielern (Clients) unterliegen einem Zeitlimit. Sowohl Clients als auch der Server warten mindestens **5** Sekunden auf Nachrichten. Wird diese Zeit überschritten, wird die Verbindung getrennt und Aktionen, die als Konsequenz einer Verbindungstrennung dienen, werden ausgeführt.

Es ist jedoch legitim, dass es innerhalb dieser Zeitspanne keine Kommunikation zwischen Server und Clients gibt, z.B. wenn die Spieler auf die nächste Runde warten. Daher sollte in regelmäßigen Abständen eine *Heartbeat*-Nachricht gesendet werden, die von der jeweiligen Gegenseite ignoriert wird, aber dafür sorgt, dass die Wartezeit zurückgesetzt wird.

Für alle Nachrichten auf dieser Seite ist angegeben, in welchen serverseitigen Modi diese zulässig sind. Entspricht der aktuelle Modus nicht den Anforderungen der Nachricht, steht es dem Spieleserver frei, diese zu ignorieren.

Folgende Kommandos und Antworten sind für den Spieleserver definiert:

Heartbeat

Bezeichner des Kommandos: `xxHeartbeat`

Gesendet von: Client, Spieleserver

Zulässig in Modi: Alle

Dieses Kommando dient ausschließlich dazu, die Verbindung zwischen Server und Clients aufrecht zu erhalten, da ansonsten die Verbindung aufgrund von Nichtaktivität getrennt werden darf. Sie transportiert keine weiteren Informationen und muss programmiertechnisch nicht weiter behandelt werden.

Sowohl Spieleserver als auch Clients müssen Heartbeats übermitteln.

Zusätzlich übermittelte Daten (`content`)

Keine.

Beispiel

Beispiel: Sende einen Heartbeat an den Client oder den Spieleserver:

```
{
    "command": "xxHeartbeat"
}
```

Einen Spieler registrieren

Bezeichner des Kommandos: `gsRegisterPlayer`

Gesendet von: Client an Spieleserver

Zulässig in Modi: Lobbymodus

Antwort: Benachrichtigung über Spielerregistrierung

Dieses Kommando assoziiert die aktuelle Verbindung zum Spieleserver mit dem angegebenen Spielernamen. Nach einer erfolgreichen Registrierung (vgl. Antwort auf dieses Kommando) kann der Spielername nicht geändert werden; der Spieler muss seine Verbindung zum Spieleserver trennen und eine neue aufbauen. Namen dürfen *ausschließlich* aus Großbuchstaben, Kleinbuchstaben und Ziffern bestehen. Tritt ein Namenskonflikt auf, wird versucht, eine laufende Nummer an den übermittelten Namen anzuhängen, um den Spielernamen eindeutig zuweisen zu können (und dem Spieler die Änderung mitgeteilt).

Zusätzlich übermittelte Daten (`content`)

- `name` (String): Name, unter dem sich der Spieler identifizieren will. Zulässige Zeichen: Großbuchstaben, Kleinbuchstaben, Ziffern nach ASCII.

Beispiele

Beispiel 1: Wir möchten uns am Spieleserver mit dem Namen `Tokyo2020` anmelden:

```
{
    "command": "gsRegisterPlayer",
    "content": {
        "name": "Tokyo2020"
    }
}
```

Der Server sendet eine Antwort, in der mitgeteilt wird, dass der Spieler unter dem Namen `Tokyo2020` oder `Tokyo20201` oder `Tokyo20202` usw. identifiziert wird.

Beispiel 2: Wir möchten uns am Spieleserver mit dem Namen `Thomas de Maizière` anmelden:

```
{
    "command": "gsRegisterPlayer",
    "content": {
        "name": "Thomas de Maizière"
    }
}
```

Der Server sendet eine Antwort, in der mitgeteilt wird, dass die Registrierung aufgrund ungültiger Zeichen fehlgeschlagen ist. Der Server muss hier das Leerzeichen sowie das kleine E mit *accent grave* (è) erkennen.

Benachrichtigung über Spielerregistrierung

Bezeichner der Antwort: `gsRRegisterPlayer`

Gesendet von: Spieleserver an Client

Zulässig in Modi: Lobbymodus

Antwort auf: Einen Spieler registrieren

Der Spieleserver sendet eine Benachrichtigung über den Registrierungsversuch an den Client zurück. Aufgrund dieser kann festgestellt werden, ob die Registrierung erfolgreich verlief oder nicht, und im Erfolgsfall zusätzlich, welchen Namen der Server dem Spieler am Ende zugewiesen hat.

Zusätzlich übermittelte Daten (`content`)

- `rejected` (Boolean): War die Registrierung erfolgreich, ist dieser Wert `false`. Wurde die Registrierung aufgrund ungültiger Zeichen abgelehnt, ist dieser Wert `true`.
- `adaptedName` (String): War die Registrierung erfolgreich, enthält diese Eigenschaft den Namen, den der Spieleserver auf Basis des vom Spieler übermittelten Namens gebildet hat. Wurde der ursprüngliche Name nicht bereits auf dem Server benutzt, ist der neue Name identisch mit dem alten, ansonsten wurde eine Nummer an den alten Namen angehängt. Wurde die Registrierung aufgrund ungültiger Zeichen abgelehnt, ist der String leer.

Beispiele

Beispiel 1: Die Registrierung des Spielers `Tokyo2020` wurde angenommen, und der Name wurde noch nicht auf dem Spieleserver verwendet:

```
{
    "command": "gsRRegisterPlayer",
    "content": {
        "rejected": false,
        "adaptedName": "Tokyo2020"
    }
}
```

Beispiel 2: Die Registrierung des Spielers `Tokyo2020` wurde angenommen, aber der Name wurde bereits auf dem Spieleserver verwendet. Jedoch ist `Tokyo20201` noch frei:

```
{
    "command": "gsRRegisterPlayer",
    "content": {
        "rejected": false,
        "adaptedName": "Tokyo20201"
    }
}
```

Beispiel 3: Die Registrierung des Spielers `Tokyo2020` wurde angenommen, aber der Name wurde bereits auf dem Spieleserver verwendet. `Tokyo20201` ist genauso belegt. Aber `Tokyo20202` ist noch frei:

```
{
    "command": "gsRRegisterPlayer",
    "content": {
        "rejected": false,
        "adaptedName": "Tokyo20202"
    }
}
```

Beispiel 3: Die Registrierung des Spielers `Thomas de Maizière` wurde aufgrund ungültiger Zeichen abgelehnt:

```
{
    "command": "gsRRegisterPlayer",
    "content": {
        "rejected": true,
        "adaptedName": ""
    }
}
```

Einen Spieler deregistrieren

Dies geschieht über das Trennen der (Socket-)Verbindung zum Spieleserver. Der Server muss auf die Trennung reagieren und einen eventuell auf der Verbindung registrierten Spieler löschen. War der getrennte Spieler in einem vergangenen Highscore vertreten, darf der Highscore-Eintrag nicht entfernt werden.

Änderung der Spielerliste

Bezeichner des Kommandos: `cPlayersUpdated`

Gesendet von: Spieleserver an Client

Zulässig in Modi: Lobbymodus

Antwort: keine

Der Spieleserver sendet bei jeder Änderung der Liste der angemeldeten Spieler an alle Clients mit angemeldeten Spielern eine Liste von Spielern, die im momentanen Zustand für die nächste Spielerunde zugelassen ist. In der Liste ist markiert, welcher Spieler Spielleiter ist. Diesem Spieler soll der Client die Konfiguration der folgenden Spielerunde anzeigen.

Die übertragene Menge wird im Spielemodus im Kommando Änderung der Spielarena mit veränderten Eigenschaften übertragen.

Zusätzlich übermittelte Daten (`content`)

- `players` (Array): Enthält eine Liste von Objekten mit angemeldeten Spielern, die für die nächste Spielerunde zugelassen sind.
  - `name` (String): Name eines Spielers, der für die nächste Runde zugelassen ist. Dieser muss unter diesem Namen auf dem Spieleserver angemeldet sein.
  - `isLeader` (Boolean): Ist auf `true` gesetzt, wenn dieser Spieler als Spielleiter ausgewählt wurde, ansonsten auf `false`. (Je nach Implementation ist für den ersten gelisteten Spieler dieser Wert stets `true` gesetzt. Dies muss jedoch nicht immer der Fall sein!)

Beispiele

Beispiel 1: Es sind momentan keine Spieler angemeldet (das Format lässt es zu, es kann aber an keine Spieler gesendet werden, da ja keine angemeldet sind):

```
{
    "command": "cPlayersUpdated",
    "content": {
        "players": []
    }
}
```

Beispiel 2: Es ist momentan ein Spieler namens `Alfredo` angemeldet. Dieser ist gleichzeitig Spielleiter:

```
{
    "command": "cPlayersUpdated",
    "content": {
        "players": [
            { "name": "Alfredo", "isLeader": true }
        ]
    }
}
```

Beispiel 3: Es sind momentan zwei Spieler namens `Alfredo` und `Bruno` angemeldet. `Alfredo` ist gleichzeitig Spielleiter:

```
{
    "command": "cPlayersUpdated",
    "content": {
        "players": [
            { "name": "Alfredo", "isLeader": true },
            { "name": "Bruno", "isLeader": false }
        ]
    }
}
```

Beispiel 4: Zehn Spieler sind angemeldet, von denen der Spieleserver jedoch nur 4 von ihnen maximal für die nächste Runde zulässt. Diese 4 stehen in der Liste, alle anderen nicht:

```
{
    "command": "cPlayersUpdated",
    "content": {
        "players": [
            { "name": "Alfredo", "isLeader": true },
            { "name": "Bruno", "isLeader": false },
            { "name": "Cristina", "isLeader": false }
            { "name": "Diana", "isLeader": false }
        ]
    }
}
```

Eine Spielarena für die nächste Runde konfigurieren

Bezeichner des Kommandos: `gsConfigureArena`

Gesendet von: Client an Spieleserver

Zulässig in Modi: Lobbymodus

Antwort: keine

Der Spielleiter ist berechtigt, während die Lobby offen ist, das Spiel zu konfigurieren. Alle Einstellungen werden gesammelt und in diesem Kommando an den Spieleserver übertragen. Ist die Konfiguration zulässig und sind mindestens zwei Spieler auf dem Server angemeldet, so startet der Spieleserver das Spiel und geht in den Spielemodus über, worauf das erste Update folgt. Folgende Einstellungen kann der Spielleiter verändern:

- Die Größe des Spielfelds (Breite und Höhe)
- Dauer der Runde in Minuten
- Dauer in Sekunden, wie lange der Effekt einer Rüstung anhält
- Dauer in Sekunden, nach welcher gelegte Bomben explodieren
- Maximale Anzahl von Spielern in der Runde
- Modus des Spiels

Zusätzlich übermittelte Daten (`content`)

- `arenaWidth` (Integer): Die Breite der Arena, in der die Runde stattfindet. Zulässig sind ungerade Werte zwischen 7 und 21 (Felder). Ein äußerer Rand um die Spielfläche wird hinzu gerechnet.
- `arenaHeight` (Integer): Die Höhe der Arena, in der die Runde stattfindet. Zulässig sind ungerade Werte zwischen 7 und 21 (Felder). Ein äußerer Rand um die Spielfläche wird hinzu gerechnet.
- `playTimeMinutes` (Integer): Anzahl der Minuten, die die Runde andauert. Zulässig sind Werte zwischen 2 und 10 (Minuten).
- `armorEffectSeconds` (Integer): Anzahl der Sekunden, wie lange der Effekt einer Rüstung anhält. Zulässig sind Werte zwischen 5 und 10 (Sekunden).
- `bombExplosionSeconds` (Integer): Anzahl der Sekunden, nach welcher gelegte Bomben explodieren werden. Zulässig sind Werte zwischen 2 und 5 (Sekunden).
- `maxPlayers` (Integer): Anzahl der Spieler, die maximal an dieser Runde teilnehmen darf. Zulässig sind Werte zwischen 2

und 4 (Spielern).

- `gameplayMode` (Integer): Legt den Modus fest, unter der die Runde stattfindet. Zulässig sind der Wert 1 (für den Modus "Standard A") oder 2 (für den Modus "Standard A+B").

Beispiel

Beispiel: Das Spielfeld ist mit Rand 13x11 Felder groß. Das Spiel dauert 6 Minuten. Die Rüstung verliert nach 7 Sekunden ihre Wirkung, während Bomben nach 3 Sekunden explodieren. Maximal nehmen 4 Spieler an der Runde teil, und es wird "Standard A+B" gespielt:

```
{
    "command": "gsConfigureArena",
    "content": {
        "arenaWidth": 13,
        "arenaHeight": 11,
        "playTimeMinutes": 6,
        "armorEffectSeconds": 7,
        "bombExplosionSeconds": 3,
        "maxPlayers": 4,
        "gameplayMode": 2
    }
}
```

Die Spielarena und/oder die Spielerliste ändert sich in der Runde

Bezeichner des Kommandos: `cArenaUpdated`

Gesendet von: Spieleserver an alle Spieler (Clients) der Runde

Zulässig in Modi: Spielmodus

Antwort: keine

Dies ist das wichtigste Kommando der gesamten Spezifikation. Nach jedem Tick muss der Spieleserver ein Update erzeugen, welches an alle Clients gesendet wird und aus welchem das Spielfeld und optional weitere Informationen gerendert werden müssen.

Es können sowohl Spieler oder einzelne Felder der Arena geändert werden oder auch beides. Ist die jeweilige Kategorie im Update leer (in JSON-Sprech ein leeres Array), so hat sich diesbezüglich nichts geändert.

Eine Änderung der Spielerliste ist notwendig, wenn sich die Position des Spielers ändert, ein Spieler eliminiert wird, oder sich der Verwundbarkeitszustand eines Spielers ändert. Es werden nur Spieler in der Liste übertragen, die nicht bereits aus der Runde eliminiert wurden. Die Reihenfolge der Spieler ist nicht definiert, da diese den konkreten Updatevorgang nicht beeinflusst.

Ändert sich die Position des Spielers, so muss diese, sofern sich nicht zwei oder mehr Spieler überlappen, durch ein Feld-Update mit der entsprechenden ID (siehe Liste) ergänzt werden. Die Arena muss ausschließlich aus Feld-Updates gezeichnet werden! Spielerupdates sind ausschließlich zur Darstellung zusätzlicher Informationen außerhalb der Arena gedacht.

Bei einer Änderung der Spielerliste wird die komplette Spielerliste neu übertragen; hier greift kein sogenanntes "Delta-Update". Änderungen der Felder in der Arena werden als "Delta-Update" übertragen; es werden nur die Felder gesendet, die sich seit dem letzten Tick verändert haben. Die Reihenfolge der Objekte ist nicht definiert, da diese den konkreten Updatevorgang nicht beeinflusst.

Die einzige Ausnahme, die kein "Delta-Update" ist, ist das allererste Update der Runde, in welchem die komplette Arena übertragen werden muss; die Größe der Arena lässt sich aus diesem Update herleiten. Anschließend sind alle weiteren Updates "Delta-Updates".

Mit dem ersten gesendeten Update müssen alle beteiligten Clients in die Arena wechseln.

Zusätzlich übermittelte Daten (`content`)

- `players` (Array): Dieses Array von Objekten ist entweder leer (wenn sich der Zustand der Spieler nicht geändert hat) oder eine Liste aller noch nicht eliminierter Spieler dieser Runde.

    - `number` (Integer): Die von der Engine zugewiesene Nummer des Spielers, entweder 1, 2, 3 oder 4.
    - `name` (String): Der Name des Spielers, so wie er sich am Server registriert hat.
    - `x` (Integer): X-Koordinate des Spielers. Diese ist minimal 0 und maximal Arenabreite minus 1.
    - `y` (Integer): Y-Koordinate des Spielers. Diese ist minimal 0 und maximal Arenahöhe minus 1.
    - `isArmored` (Boolean): Dieser Wert ist `true`, wenn der Spieler gerade eine Rüstung trägt, und ansonsten `false`.
- `updatedFields` (Array): Dieses Array von Objekten ist entweder leer (wenn sich in der Arena nichts verändert hat) oder eine Liste von Elementen, die in der Arena geändert wurden.
    - `x` (Integer): X-Koordinate des geänderten Feldes. Diese ist minimal 0 und maximal Arenabreite minus 1.
    - `y` (Integer): Y-Koordinate des geänderten Feldes. Diese ist minimal 0 und maximal Arenahöhe minus 1.
    - `id` (Integer): Die ID des Objektes, welches sich auf dem geänderten Feld befindet. Dabei darf es sich ausschließlich um die IDs aus der Liste handeln!

Beispiele

Beispiel 1: Spieler "Daniel" ist aus dem Spiel geflogen (muss noch nicht unbedingt durch Arenaupdate reflektiert werden). Ansonsten keine Änderungen. Es folgt vollständiges Spielerupdate, ohne Arenaupdate:

```
{
    "command": "cArenaUpdated",
    "content": {
        "players": [
            { "number": 1,
              "name": "Ariana",
              "x": 7,
              "y": 5,
              "isArmored": false },
            { "number": 2,
              "name": "Bernd",
              "x": 11,
              "y": 9,
              "isArmored": true },
            { "number": 3,
              "name": "Conrad",
              "x": 3,
              "y": 3,
              "isArmored": true },
        ],
        "updatedFields": []
    }
}
```

Beispiel 2: Bernd (Spieler 2) und Conrad (Spieler 3) haben im gleichen Tick vor Kurzem eine Bombe gelegt, die in Zustand 2 übergeht. Gleichzeitig wird der Explosionseffekt einer anderen Bombe aufgeräumt, wobei an den Koordinaten X=7 und Y=9 ein Powerup "Rüstung" gelegt wurde:

```
{
    "command": "cArenaUpdated",
    "content": {
        "players": [],
        "updatedFields": [
            { "x": 5, "y":  9, "id": 72 },
            { "x": 1, "y":  1, "id": 82 },
            { "x": 7, "y": 10, "id":  0 },
            { "x": 6, "y": 10, "id":  0 },
            { "x": 8, "y": 10, "id":  0 },
            { "x": 7, "y":  9, "id": 23 }
        ]
    }
}
```

Beispiel 3: Spieler "Conrad" war vorher unverwundbar, jetzt nicht mehr. Es folgt vollständiges Spielerupdate, gefolgt von einer Änderung der ID an der Position, wo Conrad steht, im Arenaupdate:

```
{
    "command": "cArenaUpdated",
    "content": {
        "players": [
            { "number": 1,
              "name": "Ariana",
              "x": 7,
              "y": 5,
              "isArmored": false },
            { "number": 2,
              "name": "Bernd",
              "x": 11,
              "y": 9,
              "isArmored": true },
            { "number": 3,
              "name": "Conrad",
              "x": 3,
              "y": 3,
              "isArmored": false },
        ],
        "updatedFields": [
            { "x":  3, "y":  3, "id": 53 }
        ]
    }
}
```


Beispiel 4: Das erste Update wird übertragen. Es enthält die komplette Spielerliste und alle Felder der Arena:

```
{
    "command": "cArenaUpdated",
    "content": {
        "players": [
            { "number": 1,
              "name": "Ariana",
              "x": 1,
              "y": 1,
              "isArmored": false },
            { "number": 2,
              "name": "Bernd",
              "x": 9,
              "y": 1,
              "isArmored": false },
            { "number": 3,
              "name": "Conrad",
              "x": 1,
              "y": 9,
              "isArmored": false },
            { "number": 4,
              "name": "Daniel",
              "x": 9,
              "y": 9,
              "isArmored": false },
        ],
        "updatedFields": [
            { "x":  0, "y":  0, "id":  1 },
            { "x":  1, "y":  0, "id":  1 },
            // ... ja, ich weiß, dass Javascript-Kommentare in JSON nicht zulässig sind ...
            { "x": 10, "y":  0, "id":  1 },
            { "x":  0, "y":  1, "id":  1 },
            { "x":  1, "y":  1, "id": 51 }, // dort steht Ariana
            { "x":  2, "y":  1, "id":  0 },
            { "x":  3, "y":  1, "id":  2 },
            // ... usw. bis zum Schluss ...
            { "x":  8, "y": 10, "id":  1 },
            { "x":  9, "y": 10, "id":  1 },
            { "x": 10, "y": 10, "id":  1 }
        ]
    }
}
```

Aus dem obigen Beispiel kann man ablesen, wie man die Größe der Arena bestimmt: Nehme die jeweils größte geänderte X- bzw. Y-Koordinate und addiere jeweils 1 dazu. Da die Felder in beliebiger Reihenfolge übertragen werden dürfen, können wir nicht annehmen, dass das letzte übertragene Feld diese Informationen enthält -- es kann auch das Objekt an den Koordinaten X=0 und Y=0 übertragen werden.

IDs der Objekte, die im Delta-Update verwendet werden

Für die, die das interne Dokument "Vorschlag JSON Format v2" kennen: Die IDs unten codieren bei Bomben "status" und "originID" mit, bei den Powerups "type". Es sind daher keine Infos verloren gegangen, sie sind bloß kompater gespeichert.

ID	Beschreibung
0	Begehbares Feld
1	Feste Wand
2	Zerstörbare Wand
3	Explosion
21	Powerup: Radius der Bombe erhöhen
22	Powerup: Mehr Bomben platzieren
23	Powerup: Rüstung
24	Powerup: Schneller laufen (A+B)
25	Powerup: Bomben treten (A+B)
26	Powerup: Super Bombe (A+B)
27	Powerup: Maximaler Bombenradius (A+B)
28	Powerup: Bombenläufer (A+B)
51	Spieler 1
52	Spieler 2
53	Spieler 3
54	Spieler 4
55	Spieler 1 (trägt Rüstung)
56	Spieler 2 (trägt Rüstung)
57	Spieler 3 (trägt Rüstung)
58	Spieler 4 (trägt Rüstung)
61	Bombe (Zustand 1, gehört Spieler 1)
62	Bombe (Zustand 2, gehört Spieler 1)
63	Bombe (Zustand 3, gehört Spieler 1)
71	Bombe (Zustand 1, gehört Spieler 2)
72	Bombe (Zustand 2, gehört Spieler 2)
73	Bombe (Zustand 3, gehört Spieler 2)
81	Bombe (Zustand 1, gehört Spieler 3)
82	Bombe (Zustand 2, gehört Spieler 3)
83	Bombe (Zustand 3, gehört Spieler 3)
91	Bombe (Zustand 1, gehört Spieler 4)
92	Bombe (Zustand 2, gehört Spieler 4)
93	Bombe (Zustand 3, gehört Spieler 4)

Mit der Spielfigur in der Runde interagieren

Bezeichner des Kommandos: `gsPlayerAction`

Gesendet von: Alle Spieler (Clients) der Runde an Spieleserver

Zulässig in Modi: Spielmodus

Antwort: keine

Die Spielfigur kann während des Spiels nach oben, nach links, nach rechts und nach unten bewegt werden. Zudem kann eine Bombe platziert werden.

Zusätzlich übermittelte Daten (`content`)

- `action` (Integer): Eine ID, die festlegt, welche Aktion durchgeführt werden soll. Zulässige Werte sind:
  - `0` platziert eine Bombe. Die Position der Bombe ist nicht durch diese Spezifikation festgelegt.
  - `1` bewegt die Spielfigur nach rechts.
  - `2` bewegt die Spielfigur nach links.
  - `3` bewegt die Spielfigur nach oben.
  - `4` bewegt die Spielfigur nach unten.

Beispiele

Beispiel 1: Die Spielfigur soll nach rechts bewegt werden:

```
{
    "command": "gsPlayerAction",
    "content": { "action": 1 }
}
```

Beispiel 2: Die Spielfigur soll nach links bewegt werden:

```
{
    "command": "gsPlayerAction",
    "content": { "action": 2 }
}
```

Beispiel 3: Die Spielfigur soll nach oben bewegt werden:

```
{
    "command": "gsPlayerAction",
    "content": { "action": 3 }
}
```

Beispiel 4: Die Spielfigur soll nach unten bewegt werden:

```
{
    "command": "gsPlayerAction",
    "content": { "action": 4 }
}
```

Beispiel 5: Die Spielfigur soll eine Bombe auf das Spielfeld platzieren:

```
{
    "command": "gsPlayerAction",
    "content": { "action": 0 }
}
```

Einen Highscore übermitteln

Bezeichner des Kommandos: `cRoundEndHighscore`

Gesendet von: Spieleserver an alle Spieler (Clients) der Runde

Zulässig in Modi: Spielmodus

Antwort: keine

Ist die Spielzeit abgelaufen oder steht nur noch 1 Spieler in der Arena, so ist die Runde zu Ende. Anschließend berechnet der Spieleserver nach nicht weiter festgelegten Regeln eine ganzzahlige (Integer-)Punktzahl für jeden Spieler und macht weiterhin Angaben (auch für jeden Spieler) über:

- Anzahl gelaufener Schritte
- Anzahl platzierter Bomben
- Anzahl zerstörter Wände
- Anzahl getöteter Spieler
- Anzahl eingesammelter Powerups
- Hat sich der Spieler mit einer eigenen Bombe getötet? (Selbstzerstörung bei Verbindungsverlust/-timeout zählt hier dazu)
- War der Spieler der letzte lebende in der Arena?

Nach Übertragung des Highscores wechselt der Spielserver zurück in den Lobbymodus. Dabei werden die bisher verbundenen Spieler in die Warteliste übernommen.

Zusätzlich übermittelte Daten (`content`)

- `players` (Array): Enthält eine Liste von Highscores pro Spieler, der in der Runde mitgespielt hat (egal ob er mittendrin getrennt wurde oder nicht). Das Array hat keine definierte Sortierung (z.B. nach dem Punktestand), diese muss von den Clients vorgenommen werden.
  - `name` (String): Name des Spielers.
  - `walkedSteps` (Integer): Anzahl der Schritte (>= 0), die der Spieler in der Runde gelaufen ist.
  - `plantedBombs` (Integer): Anzahl der Bomben (>= 0), die der Spieler in der Runde gelegt hat.
  - `destroyedWalls` (Integer): Anzahl der zerstörbaren Wände (>= 0), die der Spieler in der Runde zerstört hat.
  - `killedPlayers` (Integer): Anzahl der Spieler (>= 0), die der Spieler in der Runde mit Bomben getötet hat.
  - `collectedPowerups` (Integer): Anzahl der Powerups (>= 0), die der Spieler eingesammelt hat.
  - `isSuicided` (Boolean): `true` , wenn sich der Spieler mit einer eigenen Bombe getötet hat oder die Selbstzerstörung bei Verbindungsverlust/-timeout eingetreten ist, ansonsten `false` .
  - `isLastPlayer` (Boolean): `true` , wenn der Spieler der letzte lebende in der Arena war, ansonsten `false` .
  - `score` (Integer): Der für den Spieler aus allen obigen Parametern berechnete Punktestand.

Beispiele

Beispiel 1: In der letzten Runde haben "Christopher" und "Hiroyuki" gegeneinander gespielt. Es ist folgendes bekannt:

- Auf dem Spielfeld stand nur noch Hiroyuki, wodurch die Runde vorzeitig beendet wurde.
- Christopher ist 70 Schritte gelaufen, hat 10 Bomben platziert, 9 Wände zerstört, keinen Spieler getötet, hat 4 Powerups eingesammelt, kam nicht selbst ums Leben, war aber auch nicht Letzter in der Arena. Dafür bekommt er 83 Punkte.
- Hiroyuki ist 76 Schritte gelaufen, hat 9 Bomben platziert, 7 Wände zerstört, 1 Spieler getötet, hat 5 Powerups eingesammelt, kam nicht selbst ums Leben, und war somit Letzter in der Arena. Dafür bekommt er 97 Punkte.

```
{
    "command": "cRoundEndHighscore",
    "content": {
        "players": [
            { "name": "Christopher",
              "walkedSteps": 70,
              "plantedBombs": 10,
              "destroyedWalls": 9,
              "killedPlayers": 0,
              "collectedPowerups": 4,
              "isSuicided": false,
              "isLastPlayer": false,
              "score": 83 },
            { "name": "Hiroyuki",
              "walkedSteps": 76,
              "plantedBombs": 9,
              "destroyedWalls": 7,
              "killedPlayers": 1,
              "collectedPowerups": 5,
              "isSuicided": false,
              "isLastPlayer": true,
              "score": 97 }
        ]
    }
}
```


Beispiel 2: In der letzten Runde haben "Roosevelt", "Hitler", "Stalin" und "Tōjō" gegeneinander gespielt. Es ist folgendes bekannt:

- Die Spielzeit wurde voll ausgereizt und Roosevelt und Stalin waren noch auf dem Feld. Damit ist keiner Letzter in der Arena.
- Roosevelt ist 213 Schritte gelaufen, hat 48 Bomben platziert, 12 Wände zerstört, 1 Spieler getötet, hat 6 Powerups eingesammelt, kam nicht selbst ums Leben. Dafür bekommt er 274 Punkte.
- Hitler ist 88 Schritte gelaufen, hat 45 Bomben platziert, 33 Wände zerstört, keinen Spieler getötet, hat 14 Powerups eingesammelt. Leider wurde er während des Spiels vom Internet getrennt, kam also selbst ums Leben. Dafür bekommt er 89 Punkte.
- Stalin ist 196 Schritte gelaufen, hat 56 Bomben platziert, 18 Wände zerstört, keinen Spieler getötet, hat 7 Powerups eingesammelt, kam nicht selbst ums Leben. Dafür bekommt er 274 Punkte.
- Tōjō ist 49 Schritte gelaufen, hat 22 Bomben platziert, 9 Wände zerstört, keinen Spieler getötet, hat 2 Powerups eingesammelt, kam nicht selbst ums Leben. Dafür bekommt er 47 Punkte.

```
{
    "command": "cRoundEndHighscore",
    "content": {
        "players": [
            { "name": "Roosevelt",
              "walkedSteps": 213,
              "plantedBombs": 48,
              "destroyedWalls": 12,
              "killedPlayers": 1,
              "collectedPowerups": 6,
              "isSuicided": false,
              "isLastPlayer": false,
              "score": 274 },
            { "name": "Hitler",
              "walkedSteps": 88,
              "plantedBombs": 45,
              "destroyedWalls": 33,
              "killedPlayers": 0,
              "collectedPowerups": 14,
              "isSuicided": true,
              "isLastPlayer": false,
              "score": 89 },
            { "name": "Stalin",
              "walkedSteps": 196,
              "plantedBombs": 56,
              "destroyedWalls": 18,
              "killedPlayers": 0,
              "collectedPowerups": 7,
              "isSuicided": false,
              "isLastPlayer": false,
              "score": 274 },
            { "name": "Tōjō",
              "walkedSteps": 49,
              "plantedBombs": 22,
              "destroyedWalls": 9,
              "killedPlayers": 0,
              "collectedPowerups": 2,
              "isSuicided": false,
              "isLastPlayer": false,
              "score": 47 }
        ]
    }
}
```