1 Server

Den Server erhaltet ihr über die CoMa-Homepage `http://www.math.tu-berlin.de/CoMa/coma2.SS10/` oder über das CoMa-Forum `http://www.math.tu-berlin.de/CoMa/forum/cgi-bin/yabb2/YaBB.pl`. Der Server besteht nur aus einer einzigen Datei, die ihr je nach System per Doppelklick oder per Kommandozeile mit `java -jar Server.jar` starten könnt. Beim Aufruf über die Kommandozeile könnt ihr einige optionale Parameter festlegen:

	Bedeutung	Standardwert
1. Parameter	Port, auf dem der Server auf Clienten wartet	4444
2. Parameter	Maximale Anzahl gleichzeitiger Verbindungen	50
3. Parameter	Passwort für die Clienten	Keins
4. Parameter	Begrüßungsnachricht des Server an die Clienten	Willkommen!

Wird als Passwort „“ angegeben, wird von den Clienten kein Passwort gefordert. Leerzeichen zu Beginn und am Ende des Passworts werden ignoriert, die Groß- & Kleinschreibung wird aber beachtet. Ein Aufruf des Servers ohne Parameter entspricht dem folgenden Aufruf:

```
java -jar Server.jar 4444 50  Willkommen!
```

1.1 Kommunikation

Die Kommunikation zwischen Server und Clienten läuft über Textnachrichten ab, die mittels TCP verschickt werden. Eine Nachricht besteht dabei aus einer Textzeile, die durch | Zeichen in einen oder mehrere Abschnitte untergliedert ist. Der erste Abschnitt bestimmt die Art der Nachricht, alle weiteren Abschnitte sind Parameter, die von der Art der Nachricht abhängen.
Der Server begrüßt einen Clienten nach der Verbindung mit

```
WELCOME | <Begrüßungsnachricht>
```

und

```
AWAITING_REGISTRATION
```

Danach kann sich der Client vorstellen und einen Namen registrieren, wie im Abschnitt *Nachrichten an den Server* beschrieben ist.
Erkennt der Server die Art einer Nachricht (d.h. den ersten Abschnitt der Nachricht) nicht, antwortet er mit

```
UNKNOWN_MESSAGE | <Erster Abschnitt>
```

Ist die Art der Nachricht zum aktuellen Zeitpunkt unpassend, antwortet der Server mit

```
MESSAGE_NOT_ALLOWED_IN_CURRENT_STATE | <Erster Abschnitt>
```

Hat die Nachricht zu wenige oder zu viele Nachrichten, schickt der Server

```
INCORRECT_NUMBER_OF_PARAMETERS | <Min> | <Max> | <Geschickt>
```

wobei `<Min>` und `<Max>` jeweils die minimal und maximal zulässige Anzahl von Parametern für diese Art von Nachricht sind und `<Geschickt>` die vom Client geschickte Anzahl.

1.2 Nachrichten an den Server

Der Server akzeptiert die folgenden Nachrichten:

- INTRODUCE
- REGISTER
- SEND_SERVER_MESSAGE
- SEND_GAME_MESSAGE
- SEND_PRIVATE_MESSAGE
- ECHO
- LIST_GAME_PLAYERS
- LIST_GAMES
- LIST_PLAYERS
- LIST_REPLAYS
- LIST_SCENARIOS
- GET_GAME
- GET_REPLAY
- GET_SCENARIO
- CREATE_GAME
- JOIN_GAME
- START_GAME
- CLOSE_CONNECTION

Die genaue Syntax und Semantik dieser Nachrichten wird auf den folgenden Seiten beschrieben.

INTRODUCE | <Clienttyp> | <Passwort>	
Parameter:	1-2
Wann zulässig:	Wenn noch nicht vorgestellt

Stellt euren Client dem Server vor. Der Parameter `<Clienttyp>` ist ein beliebiger String, der euren Clienten beschreibt, z.B. `Client der Gruppe 107`. Es können mehrere Clienten des selben Typs gleichzeitig mit dem Server verbunden sein. Der Parameter `<Passwort>` ist optional und muss nur angegeben werden, wenn der Zugang zum Server durch ein Passwort geschützt ist. Das Passwort muss die korrekte Groß- & Kleinschreibung aufweisen, Leerzeichen am Anfang und Ende des Passwortes werden ignoriert. Der Server sendet

```
INTRODUCTION_SUCCESSFUL
```

wenn die Vorstellung erfolgreich verlaufen ist, anderenfalls sendet er:

```
SERVER_ACCESS_DENIED
```

Dieser Fall tritt nur dann ein, wenn der Server passwortgeschützt ist und ein falsches Passwort angegeben wurde.

Beispiel:

```
INTRODUCE | Client der Gruppe 001
INTRODUCE | Client X | Furchtbar geheimes Passwort
```

REGISTER | <Spielername>	
Parameter:	1
Wann zulässig:	Nach der Vorstellung, solange noch kein Name registriert wurde

Registriert beim Server einen Namen für euren Clienten, der vom Server benutzt wird, euch zu identifizieren. Andere Clienten können euch mit Hilfe dieses Namens Nachrichten schicken, wenn sie mit demselben Server verbunden sind. Dieser Name muss innerhalb des Servers eindeutig sein; benutzt ein anderer Client bereits den von euch gewünschten Namen, antwortet der Server mit

```
NAME_ALREADY_IN_USE
```

Wird der Name noch nicht benutzt, antwortet der Server mit

```
REGISTRATION_SUCCESSFUL
```

Eine Liste aktuell schon vergegeben Namen könnt ihr mit `LIST_PLAYERS` erhalten.

Beispiel:

```
REGISTER | Marvin
```

CLOSE_CONNECTION	
Parameter:	Keine
Wann zulässig:	Immer

Trennt die Verbindung mit dem Server. Nimmt der Client gerade an einem Spiel teil, verlässt der Client das Spiel und die Spieler erhalten eine Nachricht der Form:
 	
`PLAYER_LEFT | <Spielername>`
 	
Der Client selbst erhält die Nachricht
 	
`CONNECTION_CLOSED | As requested by client.`
 	
Der vom Client benutzte Name wird vom Server wieder zur Benutzung freigegeben.	

Beispiel:	`CLOSE_CONNECTION`

SEND_SERVER_MESSAGE | <Nachricht>	
Parameter:	1
Wann zulässig:	Sobald ein Name registriert wurde
Schickt die spezifizierte Nachricht an alle auf dem Server registrierten Spieler. Der Server schickt sie in der Form `SERVER_CHAT_MESSAGE | <Sender> | <Nachricht>` an alle Spieler weiter.	
Beispiel:	`SEND_SERVER_MESSAGE | Hallo Welt!`

SEND_GAME_MESSAGE | <Nachricht>	
Parameter:	1
Wann zulässig:	Wenn man ein Spiel spielt
Schickt die spezifizierte Nachricht an alle Spieler, die gerade mit dem Client spielen. Der Server schickt sie in der Form `GAME_CHAT_MESSAGE | <Sender> | <Nachricht>` an diese Spieler weiter.	
Beispiel:	`SEND_GAME_MESSAGE | Hallo Welt!`

SEND_PRIVATE_MESSAGE | <Spielername> | <Nachricht>	
Parameter:	2
Wann zulässig:	Wenn man auf dem Server registriert ist

Schickt die spezifizierte Nachricht an den Spieler mit dem angegebenen Namen. Ist auf dem Server kein Spieler mit diesem Namen registriert, antwortet der Server mit
`PLAYER_NOT_FOUND`
Anderenfalls leitet der Server die Nachricht in der Form
`PRIVATE_CHAT_MESSAGE | <Sender> | <Empfänger> | <Nachricht>`
an Sender und Emfänger weiter.	

Beispiel:	`SEND_PRIVATE_MESSAGE | Marvin | Don't panic!`

ECHO | <Nachricht>	
Parameter:	1
Wann zulässig:	Wenn man auf dem Server registriert ist

Schickt die spezifizierte Nachricht an sich selbst. Dieser Befehl ist eine Abkürzung für
`SEND_PRIVATE_MESSAGE | <Sender> | <Nachricht>`
entsprechend schickt der Server
`PRIVATE_CHAT_MESSAGE | <Sender> | <Sender> | <Nachricht>`
zurück.	

Beispiel:	`ECHO | Echooooooo`

LIST_PLAYERS	
Parameter:	0
Wann zulässig:	Wenn man auf dem Server vorgestellt ist
Fragt eine Liste der aktuell auf dem Server registrierten Spieler ab. Der Server schickt sie in der Form `PLAYERS | <Spieler1> | <Spieler2> | ...` an den Client. `SpielerX` ist dabei der Name eines Spielers.	
Beispiel:	`LIST_PLAYERS`

LIST_GAMES	
Parameter:	0
Wann zulässig:	Wenn man auf dem Server registriert ist
Fragt eine Liste der aktuell auf dem Server existierenden Spiele ab. Der Server schickt sie in der Form `GAMES | <Spiel1> | <Spiel2> | ...` an den Client. `SpielX` ist dabei der Name eines Spiels.	
Beispiel:	`LIST_GAMES`

LIST_SCENARIOS	
Parameter:	0
Wann zulässig:	Wenn man auf dem Server registriert ist
Fragt eine Liste der aktuell auf dem Server verfügbaren Szenarien ab, aus denen Spiele erstellt werden können. Der Server schickt sie in der Form `SCENARIO | <Szenario1> | <Szenario2> | ...` an den Client. `SzenarioX` ist dabei der Name eines Szenarios.	
Beispiel:	`LIST_SCENARIOS`

LIST_REPLAYS	
Parameter:	0
Wann zulässig:	Wenn man auf dem Server registriert ist
Fragt eine Liste der aktuell auf dem Server gespeicherten Replays ab. Ein Replay stellt dabei ein Protokoll eines auf dem Server geführten Spiels dar. Replays werden vom Server automatisch nach Spielende erzeugt, sofern ein Spiel nicht vorzeitig beendet wurde. `REPLAYS | <Replay1> | <Replay2> | ...` an den Client. `ReplayX` ist dabei der Name eines Replays, welcher sich aus dem Name des Spiels und Datum & Zeit, zu der das Spiel beendet wurde, zusammensetzt.	
Beispiel:	`LIST_REPLAYS`

LIST_GAME_PLAYERS | <Spielname>	
Parameter:	1
Wann zulässig:	Wenn man auf dem Server registriert ist

Fragt eine Liste der Spieler ab, die dem angegebenen Spiel beigetreten sind. Existiert kein Spiel mit dem angegebenen Namen, antwortet der Server mit

```
GAME_NOT_FOUND | <Spielname>
```

Anderenfalls wird die Liste in der Form

```
GAME_PLAYERS | <Spieler1> | <Spieler2> | ...
```

an den Client geschickt. `SpielerX` ist dabei der Name eines Spielers.

Beispiel: `LIST_GAME_PLAYERS | CoMa-Testspiel`

GET_GAME | <Spielname>	
Parameter:	1
Wann zulässig:	Wenn man auf dem Server registriert ist

Fragt ein auf dem Server existierendes Spiel ab. Existiert kein Spiel mit dem angegebenen Namen, antwortet der Server mit

```
GAME_NOT_FOUND | <Spielname>
```

Anderenfalls werden Informationen über das Spiel in der Form

```
GAME | <Szenario> | <Akt. # Spieler> | <Max. # Spieler> | <Laufend?>
```

geschickt. `<Szenario>` ist dabei der Name des Szenarios, auf dem das Spiel basiert, `<Akt. # Spieler>` und `<Max. # Spieler>` sind die aktuelle und die maximale Anzahl Spieler für das Spiel und `<Laufend?>` gibt an, ob man dem Spiel noch beitreten kann oder nicht.

Beispiel: `GET_GAME | CoMa-Testspiel`

GET_REPLAY | <Replayname>	
Parameter:	1
Wann zulässig:	Wenn man auf dem Server registriert ist

Fragt ein auf dem Server gespeichertes Replay ab. Existiert das gewünschte Replay nicht, antwortet der Server mit

```
REPLAY_NOT_FOUND | <Replayname>
```

Anderenfalls wird das Replay in der Form

```
REPLAYS | <Replay-Daten>
```

an den Client geschickt, wo bei die `Replay-Daten` alle vom Server geschickten Spiel-Nachrichten sind, jeweils durch `||` getrennt.

Beispiel: `GET_REPLAY | Blubb (28.04.10 12:30:18)`

CREATE_GAME | <Szenarioname> | <Spielname>	
Parameter:	2
Wann zulässig:	Wenn man auf dem Server registriert ist und gerade nicht spielt

Erzeugt ein neues Spiel mit dem angegebenen Namen basierend auf dem angegebenen Szenario. Der Spielname muss serverweit eindeutig sein, wird der Name schon benutzt, antwortet der Server mit

```
NAME_ALREADY_IN_USE | <Spielname>
```

Existiert das angegebene Szenario nicht, wird

```
SCENARIO_NOT_FOUND | <Szenarioname>
```

zurückgegeben. Anderenfalls tritt der Client dem erzeugten Spiel bei und es wird an alle registrierten Spieler

```
GAME_CREATED | <Spielname>
```

geschickt.

Beispiel: `CREATE_GAME | Risky Exchange | CoMa-Testspiel`

JOIN_GAME | <Spielname>	
Parameter:	1
Wann zulässig:	Wenn man auf dem Server registriert ist und gerade nicht spielt

Tritt dem Spiel bei, sofern noch Plätze frei sind und das Spiel nicht bereits läuft. Findet der Server kein Spiel mit dem angegebenen Namen, wird

```
GAME_NOT_FOUND | <Spielname>
```

zurückgegeben. Läuft das Spiel bereits, oder sind alle Plätze besetzt, antwortet der Server mit

```
JOINING_FAILED | <Spielname>
```

Anderenfalls erhalten der Client und alle weiteren Spieler, die dem Spiel bisher beigetreten sind, die Nachricht

```
PLAYER_JOINED | <Spielername>
```

Sollte ein Spieler zu einem späteren Zeitpunkt ein Spiel verlassen, wird

```
PLAYER_LEFT | <Spielername>
```

geschickt.

Beispiel: `JOIN_GAME | CoMa-Testspiel`

START_GAME	
Parameter:	0
Wann zulässig:	Wenn man auf dem Server registriert ist und ein Spiel erstellt hat

Startet ein Spiel, was man zuvor erstellt hat. Weitere Spieler können dem Spiel dann nicht mehr beitreten. Läuft das Spiel bereits, anwortet der Server mit

```
GAME_IS_ALREADY_RUNNING
```

Der Server fängt dann mit dem Spiel an und schickt an alle mitspielenden Clienten folgende Nachrichten:

```
GAME_STARTED | <Spielname>
```

Anschließend wird die Reihenfolge der Spieler zufällig ausgelost, die resultierende Reihenfolge wird den Spielern mittels Nachrichten der Form

```
GAME_PLAYERS | <Spieler1> | <Spieler2> | ...
```

bekannt gegeben. Danach erhalten alle Clienten die Szenario-Daten mittels einer

```
GAME_STATUS | ...
```

Nachricht. Die Scenario-Daten haben dieselbe Form wie bei `GET_SCENARIO`.

Beispiel: `START_GAME`

GAME_CHOICE | <Art der Auswahl> | <Auswahl>
Parameter: 2
Wann zulässig: Wenn man ein Spiel spielt

Trifft eine vom Server mit

```
CHOOSE | <Art der Auswahl> | <Anzahl> | <Option1> | <Option2> | ...
```

angeforderte Auswahl. `<Art der Auswahl>` muss in dieser Nachricht identisch zu der in der `CHOOSE`-Nachricht sein. `<Auswahl>` besteht aus einer durch Kommata getrennten Liste von Zahlen. Die Anzahl der Zahlen in dieser Liste muss der `<Anzahl>` aus der `CHOOSE`-Nachricht entsprechen. Die Zahlen werden als Indizes der in der Server-Nachricht gelisteten Optionen interpretiert, wobei 0 der Index der ersten Option ist. Dementsprechend dürfen die Zahlen in der Auswahlliste nur ganze Zahlen zwischen 0 und # Optionen-1 sein; außerdem darf keine Zahl doppelt vorkommen. Der Server gibt die Wahl des Spielers mittels

```
CHOSEN | <Spielername> | <Art der Auswahl> | <Gewählte Objekte>
```

bekannt. Trifft ein Client eine ungültige Wahl, ignoriert der Server diese und wählt stattdessen zufällig. Antwortet der Client nicht innerhalb einer serverspezifischen Frist auf eine `CHOOSE`-Nachricht, trifft der Server eine zufällige Auswahl und sendet

```
TIMEOUT | <Spielername> | <Art der Auswahl>
```

an alle Spieler. Erkennt der Server die vom Client angegebene `<Art der Auswahl>` nicht, antwortet er mit

```
UNKNOWN_CHOICE
```

Erkennt der Server die Art der Auswahl und erwartet sie zu dem aktuellen Zeitpunkt aber nicht, schickt er

```
NOT_WAITING_FOR_THIS_CHOICE
```

zurück. Enthält `<Auswahl>` etwas ungültiges (z.B. Zeichenketten), wird

```
ILLEGAL_CHOICE
```

zurückgegeben.

Beispiel: `GAME_CHOICE | PROGRAMMING | 0,8,3,4,5`

GET_SCENARIO | <Szenarioname>	
Parameter:	1
Wann zulässig:	Wenn man auf dem Server registriert ist

Fragt ein auf dem Server existierendes Szenario ab. Existiert kein Szenario mit dem angegebenen Namen, antwortet der Server mit

```
SCENARIO_NOT_FOUND | <Szenarioname>
```

Anderenfalls werden Informationen über das Szenario in der Form

```
SCENARIO | <Name> | <Breite> | <Höhe> | <Schwierigkeit> | <Dauer>
| <Min. Spieler> | <Max. Spieler> | <Autor> | <Beschreibung>
| <Spielfeld>
```

geschickt. `<Name>` ist dabei der Name des Szenarios, `<Breite>` und `<Höhe>` sind Breite und Höhe des (immer rechteckigen) Spielfelds. `<Schwierigkeit>` ist ein vom Author festgelegter Hinweis auf die Schwierigkeit der Strecke (`EASY`, `MEDIUM` oder `EXPERT`); `<Dauer>` ein Hinweis auf die zum Spielen des Szenarios benötigte Zeit (`SHORT`, `MEDIUM` oder `LONG`). `Min. Spieler` und `Max. Spieler` geben die empfohlende Mindestanzahl von Spielern und die maximale Anzahl von Spielern für das Szenario an, es wird nur letztere vom Server erzwungen. `<Autor>` ist der Ersteller der Karte, `<Beschreibung>` ist eine vom Autor verfasste Beschreibung der Karte. `<Spielfeld>` beschreibt schließlich das Spielfeld, siehe dazu den nächsten Abschnitt.

Beispiel:	`GET_SCENARIO | Risky Exchange`

1.3 Beschreibung des Spielfelds

Jede Zeile des rechteckigen Spielfelds wird durch ein | terminiert.
Ein Feld des Spielplans besteht aus einem oder mehreren Fabrikelementen, die sich entweder in der Mitte des Feldes oder an einem der vier Ränder (Norden, Osten, Süden, Westen) des Feldes befinden können. Ein Fabrikelement wird durch eine Folge von Großbuchstaben kodiert, die die Art des Fabrikelements angeben. An die Großbuchstaben angehängte Kleinbuchstaben und Zahlen dienen als Parameter für das Fabrikelement.
An Phasen-aktive Elemente können die Ziffern 1-5 angehängt werden um zu beschreiben, in welchen Phasen diese Elemente aktiv sind. `PU24` steht z.B. für einen Pusher, der in der 2. und 4. Phase aktiv wird. Band-Elemente wie Förderbänder haben eine Richtung, in der sie Transporieren, welche durch einen Kleinbuchstaben (n,e,s,w für Norden, Osten, Süden, Westen) direkt nach den Großbuchstaben angegeben wird. Werden keine weiteren Kleinbuchstaben angegeben, wird davon ausgegangen, dass das Band aus der gegenüberliegenden Richtung kommt. Ansonsten können die Herkunftsrichtungen durch weitere Kleinbuchstaben beschrieben werden. `Cwns` ist z.B. ein Förderband, was von Norden und Süden nach Westen transportiert (ein T-Stück also). Die Ziffern der Checkpoints definieren die Reihenfolge, in der die Checkpoints angefahren werden müssen, der erste Checkpoint hat dabei die Nummer 1. Eine Übersicht der Elemente findet sich in der Tabelle.
Besitzt ein Feld mehrere Elemente an derselben Position, z.B. eine Mauer und ein Laser, werden die Elemente durch Leerzeichen getrennt und mit runden Klammern eingeschlos-

Abk.	Zugehöriges Element	Band	Phasen	Rand	Weitere Parameter
_	Leer	Nein	Nein	Überall	–
C	Förderband	Ja	Nein	Mitte	–
CR	Presse	Nein	Ja	Mitte	–
CP	Checkpoint	Nein	Nein	Mitte	Ziffer von 1-# Checkpoints
E	Expressförderband	Ja	Nein	Mitte	–
G	Zahnrad	Nein	Nein	Mitte	l oder r für die Richtung
L	Laser	Nein	Nein	Rand	–
P	Grube	Nein	Nein	Mitte	–
PU	Schieber	Nein	Ja	Rand	–
R	Reparaturfeld	Nein	Nein	Mitte	–
SP	Startpunkt	Nein	Nein	Mitte	Ziffer von 1-# Spieler
U	Großes Reparaturfeld	Nein	Nein	Mitte	–
W	Mauer	Nein	Nein	Rand	–

sen, in dem Beispiel so: `(W L)`.
Die unterschiedlichen Positionen eines Feldes (Mitte, Norden, Osten, Süden, Westen) werden wie folgt dargestellt: zuerst die Mitte, dann die vier Ränder in der Reihenfolge Norden, Osten, Süden, Westen eingeschlossen in eckige Klammern. Ein Feld mit Mauer und Laser in Norden und Westen sowie einer in der ersten Phase aktiven Presse wäre: `CR1[(W L)__(W L)]`. Ein leeres Feld wäre entsprechend `_[____]`. Für Felder mit leerem Rand gibt es eine Abkürzung – fehlen die eckigen Klammern nach einem Feld, wird automatisch ein leerer Rand angenommen.
Felder in einer Zeile werden durch Leerzeichen getrennt.

1.4 Kommunikation während des Spiels

Nachdem die Clienten die `GAME_STATUS`-Nachrichten erhalten haben, werden sie der Reihe nach mit `CHOOSE | SPAWN_DIRECTION` aufgefordert, die Startrichtung für ihren Roboter zu wählen.
Zu Beginn einer neuen Runde erhalten die Spieler eine `NEW_TURN | <Rundennummer>`-Nachricht; abgeschaltete Roboter werden mittels `CHOOSE | REMAIN_POWERED_DOWN` gefragt, ob sie abgeschaltet bleiben wollen. Anschließend erhalten die Roboter ihre Karten und werden per `CHOOSE | PROGRAMMING` aufgefordert, ihre Programmierung vorzunehmen, was alle Roboter parallel machen. Nachdem die Programmierung abgeschlossen ist, werden die Roboter der Reihe nach per `CHOOSE | ANNOUNCE_POWER_DOWN` gefragt, ob sie sich in der nächsten Runde abschalten wollen.
Danach sendet der Server `EXECUTING_PROGRAM_CARDS` und fängt an, die Programme auszuführen. Vor Beginn jeder Phase sendet er dabei jeweils `NEW_PHASE | <Phasennummer>` an die Spieler. Unmittelbar vor dem Ausführen einer Programm-Karte kündigt der Server mittels `EXECUTE_PROGRAM_CARD | <Programm-Karte>` an, was gerade ausgeführt wird. Am Ende jeder Phase und am Ende jeder Runde erhalten alle Spieler den aktuellen Zu-

stand jedes Roboters mittels einer Nachricht der Form:

```
ROBOT_STATUS | <Name> | <Lebenspunkte> | <Max. Lebenspunkte>
| <Zerstört?> | <Fortschritt> | <Archiv-Feld> | <Aktuelles Feld>
| <Richtung> | <Abschalten angekündigt?> | <Abgeschaltet?>
```

Dabei ist `<Name>` der Name des Spieler, dem der Roboter gehört; `<Lebenspunkte>` und `<Max. Lebenspunkte>` sind die aktuellen und maximalen Lebenspunkte des Roboters. `<Zerstört?>` gibt an, ob der Roboter zerstört ist, `<Fortschritt>` gibt die Anzahl erreichter Checkpoints an. `<Archiv-Feld>` und `<Aktuelles Feld>` geben die Position der Sicherungskopie und des Roboters an, die Koordinaten haben die Form (x, y), wobei der Ursprung das oberste linke Feld des Spielplans ist. `<Richtung>` gibt die Richtung des Roboters an (NORTH, EAST, SOUTH, WEST), `<Abschalten angekündigt?>` und `<Abgeschaltet?>` genau das, was sie vermuten lassen.
Wird ein Spieler zerstört (d.h. sein Roboter ist zerstört und er hat keine Leben mehr), wird `PLAYER_DESTROYED | <Spielername>` gesendet. Erreicht ein Spieler den letzten Checkpoint, wird `PLAYER_ARRIVED | <Spielername>` geschickt. Sobald alle Spieler zerstört oder angekommen sind ist das Spiel vorbei und es wird `GAME_OVER | <Spieler1> | <Spieler2> | ...` gesendet, wobei die Spieler in ihrer Ankunfsreihenfolge geordnet sind.