**AudioRacer**

**Server Dokumentation**

Inhalt

[1) Packages und Klassen 2](#_Toc371284687)

[Projekt Server 2](#_Toc371284688)

[Projekt Core 2](#_Toc371284689)

[Projekt Spieler Kommunikation 2](#_Toc371284690)

[Projekt Welt Kommunikation 2](#_Toc371284691)

[2) Kommunikationsaufbau 2](#_Toc371284692)

[Spieler-Server Kommunikationsaufbau 2](#_Toc371284693)

[Kamera-Server Kommunikationsaufbau 2](#_Toc371284694)

[ZigBee-Server Kommunikationsaufbau 3](#_Toc371284695)

[3) Spielvorbereitung 3](#_Toc371284696)

[4) Schnittstellenbeschreibungen 4](#_Toc371284697)

[5) Offenes 5](#_Toc371284698)

[6) Notizen 5](#_Toc371284699)

# 1) Packages und Klassen

Dies sind die Packages und Klassen welche für diese Dokumentation von Bedeutung sind.

## Projekt Server

at.fhv.audioracer.server.CameraServer

at.fhv.audioracer.server.PlayerServer (Singleton)

at.fhv.audioracer.server.CarClientManager (Singleton)

at.fhv.audioracer.server.dao.ServerPlayer

at.fhv.audioracer.server.dao.ServerCar

at.fhv.audioracer.server.game.GameModerator (Singleton)

at.fhv.audioracer.server.game.Ranking

## Projekt Core

at.fhv.audioracer.core.model.Player

at.fhv.audioracer.core.model.Car

## Projekt Spieler Kommunikation

at.fhv.audioracer.communication.player.IPlayerServer (vorher: IPlayerManager)

at.fhv.audioracer.communication.player.dao.CheckPointDirection

## Projekt Welt Kommunikation

at.fhv.audioracer.communication.world.ICarClientManager

at.fhv.audioracer.communication.world.ICamera

at.fhv.audioracer.communication.world.dao.CarPosition

# 2) Kommunikationsaufbau

## Spieler-Server Kommunikationsaufbau

Die Spieler bauen zum Server eine TCP und UDP Verbindung auf.

Der Server hört auf PlayerNetwork.PLAYER\_SERVICE\_PORT und für jede neue Spieler Verbindung wird eine neue com.esotericsoftware.kryonet.Connection geöffnet.

Beim Verbindungsaufbau wird der Spieler **global im GameModerator angemeldet** (=Verwalter) und er vergibt eine ID für den Spieler.

Die „connect“-Methode (IPlayerServer) weist dem Spieler anschließend den gewünschten Namen zu.

## Kamera-Server Kommunikationsaufbau

Die Kamera verbindet sicher über TCP und UDP zum Server (CameraServer extends com.esotericsoftware.kryonet.Server).

Der Server hört auf WorldNetwork.CAMERA\_SERVICE\_PORT.

Der CameraServer lässt genau eine Kamera-Verbindung zu. Das registrierte RemoteObjekt implementiert ICamera. Über dieses können: „configureMap“, „carDetected“ und „detectionFinished“ aufgerufen werden.

Zur Übermittlung der UDP Message „updateCar“ wird das CarPosition DAO verwendet. (Siehe Listener Beispiele: <https://code.google.com/p/kryonet/>)

**Der CameraServer unterhält sich Server-intern ausschließlich mit dem GameModerator.**

## ZigBee-Server Kommunikationsaufbau

Meldet sich am ZigBee-Router (nicht Teil des Servers) ein neues ZigBee-Modul an, so meldet dies der Router über die „connect“-Methode des CarClientMangers an den Server. CarClientManger implementiert ICarClientManager.

Die ZigBee-Router Applikation wird in der gleichen VM ausgeführt und hat somit Zugriff auf CarClientManager.

Die Objekte vom Typ ICarClient welche an „connect“ übergeben werden, werden **global im GameModerator** **angemeldet**.

**ICarClientManager wird nebenläufig verwendet. Wird in dessen Methoden zum Beispiel GameModerator Methoden zugegriffen muss hier auf diese Nebenläufigkeit geachtet werden. Wenn nicht in dieser Klasse, dann zumindest im GameModerator selber.**

# 3) Spielvorbereitung

Voraussetzungen für Spiel Start:

* der CameraServer muss „configureMap“ zumindest einmal aufgerufen haben
* der CamerServer muss „detectionFinished“ mindestens einmal aufgerufen haben
* der GameModerator muss die Verbindung zwischen allen bekannten ICarClients und allen bekannten ServerCars hergestellt haben.

Dies geschieht nachdem „detectionFinished“ an den GameModerator gemeldet wurde.

**Diese ist der Idealfall dieser Voraussetzung.**

* der GameModerator muss alle bekannten ServerCars mit einem ServerPlayer in Verbindung gebracht haben.

**Diese ist der Idealfall dieser Voraussetzung.**

* Alle Spieler welche mit einem ServerCar in Verbindung gebracht worden sind, müssen über „setPlayerReady“ ihres RemoteObjectes deren Bereitschaft signalisiert haben.

„configureMap“ kann nur aufgerufen werden, wenn das Spiel nicht läuft.

„detectionFinished“ kann nur aufgerufen werden, wenn das Spiel nicht läuft und zumindest ein ServerCar von „carDetected“ gemeldet wurde.

„carDetected“ kann nur aufgerufen werden, wenn das Spiel nicht läuft und „detectionFinished“ noch nicht aufgerufen wurde.

Das obige wird in einer Schleife immer wieder überprüft. Sind alle Vorbedingungen erfüllt, initialisiert das Ranking Objekt die Spieler der Autos (verbleibende Checkpoints, Time auf eine einheitliche Startzeit) und berechnet die Wegpunkte.

Sind die Vorbereitungen des Ranking Objektes abgeschlossen, startet das Spiel.

Anmerkungen:

ZigBee „connect“ als auch „disconnect“ des ICarClientManagers soll auch während des Spielgeschehens möglich sein. Da jedoch noch nicht genau klar ist, wie der Server dies handhaben soll, wird dies zurzeit, während ein Spiel läuft nicht funktionieren.

# 4) Schnittstellenbeschreibungen

**„updateCar**

*während des Spiels*

Der GameModerator bekommt vom CameraServer über „updateCar“ Bewegungsinformation der Autos mitgeteilt. Dieser schaut nach, wie viele \_coinsLeft der Spieler dieses Autos hat und delegiert anschließend die aktuellen Werte (carId, Position und Direction) an sein Ranking Objekt weiter. Das Ranking Objekt berechnet die aktuelle CheckPointDirection und gibt dies als Rückgabewert retour. Das Ranking Objekt bestimmt zusätzlich ob der Spieler einen Checkpoint während dieses update-Prozesses erreicht hat und dekrementiert gegebenenfalls auch den \_coinsLeft Counter des Spielers. Nachdem der GameModerator die aktuelle CheckPointDirection des Autos bekommen hat, sendet er per UDP „updateCheckpointDirection“ an den Spieler des Autos und schaut nach ob \_coinsLeft vom Ranking Objekt dekrementiert wurde. Ist dies der Fall, benachrichtigt der GameModerator alle Spieler über TCP „updateCheckPointDirection“.

Wenn \_coinsLeft eines Spielers 0 erreicht hat, wird die CheckPointDirection auf den letzten Wegpunkt zeigen und kann somit auch (0,0) betragen.

**„updateFreeCars“**

Meldet die Camera während der Spielvorbereitung „carDetected“ an den GameModerator, so sendet dieser per PlayerServer.sentToAllTCP „updateFreeCars(int[])“ an alle Spieler.  
„disconnect“ während des Spiels und in der Spielvorbereitung löst ebenfalls solch eine Benachrichtigung aus.

**„selectCar“**

Sendet ein Spieler „selectCar“, wird der Spieler mit dem Auto in Verbindung gebracht. Der GameModerator sendet PlayerServer.sendToAllExceptTCP „playerConnected“ und PlayerServer.sentToAllTCP „updateFreeCars“ und antwortet dem Spieler mit true, falls die Verbindung erfolgreich hergestellt werden konnte. War ein anderer Spieler schneller oder hat der Spieler sich über „connect“ keinen Namen zugeteilt, wird false retourniert. „selectCar“ Aufrufe sind nur in der Spielvorbereitung möglich.

**„playerConnected“**

Dies wird vom GameModerator mit PlayerServer.sendToAllExceptTCP an alle Spieler (außer den Spieler, den diese Meldung primär betrifft) geschickt wenn ein Spieler sich über „selectCar“ mit einem Auto verbunden hat. „playerConnected“ ist somit nur in der Spielvorbereitung von Bedeutung.

**„disconnect“, „playerDisconnected“**

Kann zu jeder Zeit an den Server gesendet werden. Der GameModerator entkoppelt den Spieler von seinem Auto. Wird „disconnect“ während einer Spielrunde aufgerufen, kann dies, so lange das Spiel läuft, nicht mehr Rückgängig gemacht werden. Nach Spiel Ende kann das Auto theoretisch über „selectCar“ wieder mit dem Spieler verbunden werden.

War die Endkoppelung erfolgreich, passiert sowohl in der Spielvorbereitung als auch während eines laufenden Spieles folgendes:

* alle anderen Spieler werden über „playerDisconnected“ (PlayerServer.sendToAllExceptTCP) informiert, dass der Spieler zur Zeit mit keinem Auto in Verbindung steht
* an alle Spieler wird „updateFreeCars“ gesendet

**„updateVelocity“**

Während eines laufenden Spieles kann der Spieler über „updateVelocity“ Steuerungskommandos an den Server schicken. Intern werden die Steuerungskommandos vom GameModerator auf das Auto, welches mit dem Spieler verbunden ist, und seine ZigBee-Verbindung (ICarClient) gemappt und an diese weitergereicht.

**„gameOver“ (TCP!)**

Entscheidet der GameModerator, dass das Spiel nun zu Ende ist, sendet er „updateVelocity(0,0)“ an alle Autos. Sendet PlayerServer.sendToAllTCP „endGame“ an alle Spieler und lässt keine „updateVelocity“ Kommandos der Spieler mehr zu.

Empfangen die Spieler-Clients „endGame“, setzen diese ihre „playerReady“ Flag wieder auf false. Dies ermöglich dem Spieler sich nun:

* vom Auto zu entkoppeln „disconnect“
* sich über „setPlayerReady“ an der nächsten Spielrunde anzumelden

**„getCarImage“**

Kann zu jeder Zeit aufgerufen werden.

# 5) Offenes

* „invalidCommand“ … Verwendung offen
* Schnittstellenbeschreibung für Spielstart offen (Server -> Player)
* ServerPlayer sollte sowohl von Player (Core Projekt) als auch Kyronet Connection erben….. \*boing\*

# 6) Notizen

* Der GameModerator muss Zugriff auf den PlayerServer haben um z.B. Broadcasts zu tätigen.
* Der GameModerator verwaltet -> die Spieler, die Autos und die ZigBee-Verbindungen. Verwaltung heißt auch Koppelung dieser Entitäten.
* ServerCar hat Property „carClient“ vom Type ICarClient.
* **Methodenaufrufe welche auf Objekten aufgerufen werden welche von com.esotericsoftware.kryonet.Connection erben, werden nebenläufig aufgerufen!**

**(und Methoden, welche diese dann aufrufen und so weiter ….)**