## Synchronisation Ticket\_To\_Ride\_Server **Threads** Welcome Game Player Socket Model Greift auf die Daten zu Player call linking sync / returns Method action call Estabilsh communication thread Parst empfangene Session Kommandos Welcom Socket put msg start game() sync / to action call sending thread Sending Thread **Receiving Thread**

Game

Thread

Nur eine Datenstruktur, kein Thread. Trotzdem immer nur **EIN ZUGRIFF VON EINEM THREAD ZUR GLEICHEN ZEIT!!!** 

Für alle attribute **EIN ZUGRIFF VON EINEM THREAD** GameModel **ZUR GLEICHEN ZEIT!!!** 

## **Ablauf eines Kommandos**

Zu Beginn wartet der Empfänger auf ein Signal vom Client. Kommt eines, wird der ReceivingThread erweckt und begibt auf die Reise durch die Klassen (hier dargestellt durch die roten Pfeile). Am Ende wirft der ReceivingThread die Antwort in die Message Queue vom SendingThread. Dieser sendet dann.

## Game Thread

Lobby

regest or change state

reqest or change state

returns

received

Sending

Thread

Sendet an den Client

command

Receiving

Thread

**Empfängt Kommandos** 

sync /

action call

Dieser (grüner Pfeil) wandert auch über die Player Klasse zum sending thread. Da es aber nur maximal einen Game Thread pro Spieler gibt und die Methoden die dieser nutzt nur von einem Game Thread aufgerufen gleichzeitig werden, muss hier im Player und abwärts nicht synchronisiert werden.

## Wo Synchronisation nötig ist

Es gibt zwei Orte, an denen mehrere Threads gemeinsam Daten nutzen und auch schreiben. Die Lobby und das Game Model. Hier ist zu synchronisieren.

Damit das übersichtlich ist, soll nicht von der Session auf die Lobby oder das GameModel zugegriffen werden, sondern nur von der Player Klasse aus.

Dabei ist sicherzustellen, dass man in den kritischen Abschnitten alleine ist.