Nico Bachmann, Vladimir Buser, Maria Desteffani, Andrina Geller

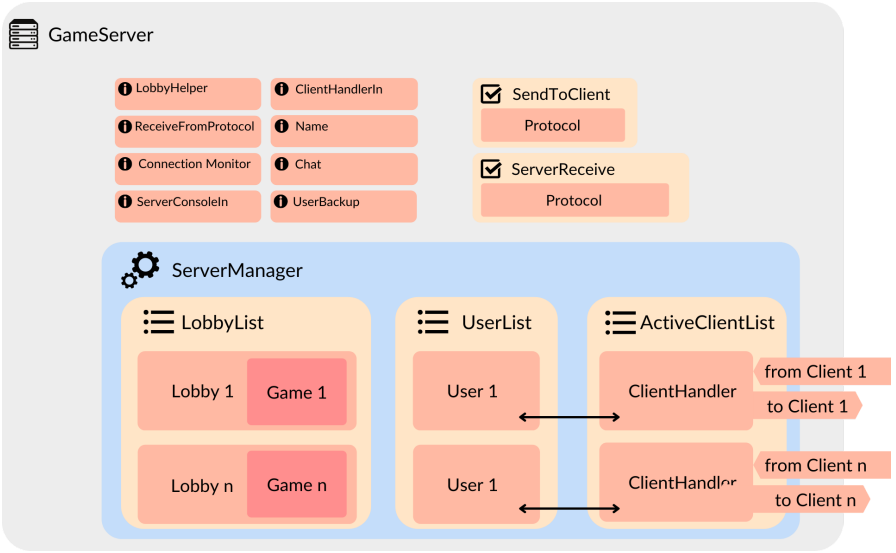
Program Architecture

The Student Games - by District 9

Das Programm (Spiel) **The Student Games** von District-9 basiert auf einer in Java implementierten Server-Client-Architektur. Ziel war es, ein Multiplayer-Spiel zu entwickeln, bei dem mehrere Clients (Spieler) zusammen auf einem Server spielen können. Dabei soll der Server stets die Kontrolle über den Spielverlauf haben, Spielzüge auf ihre Richtigkeit überprüfen und die Anfragen der Clients korrekt beantworten.

Der Grundstein unseres Spiels war der Code des Echo-Servers, welcher in der Vorlesung als mögliches Beispiel für eine Server-Client Architektur vorgestellt wurde. Wir haben diesen Code dann an unsere Bedürfnisse angepasst und entsprechend erweitert, bis am Ende ein funktionierendes Multiplayer-Spiel entstanden ist.

Die Serverseite:



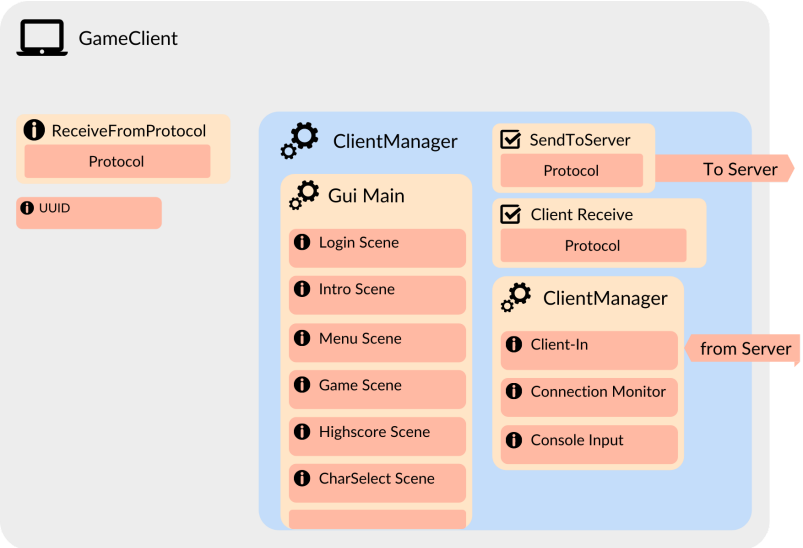
Wir haben uns bemüht, die benötigten Funktionen des Servers möglichst effizient zu unterteilen und eine Struktur zu haben, bei der wir nicht die Übersicht verlieren.
Der **GameServer** ist die Klasse, die vom Starter aufgerufen wird und den Server startet. Nach dem Start empfängt sie neue Verbindungen und leitet diese an den **ServerManager** weiter.

Der **ServerManager** hat den Überblick über alles, was auf dem Server passiert.
Er hat eine Liste mit allen aktiven Verbindungen, mit allen bisher erstellen Spielern und mit allen Lobbies und Spielen.
Da der **ServerManager** static ist, können auch alle Klassen im Server unkompliziert Informationen abfragen, welche der **ServerManager** kennt.

Der wichtigste Teil ist aber eindeutig der **ClientHandler**, welcher jeweils die Verbindung zu je einem Client hält. Er besitzt die In/Out-Streams und beinhaltet alles, was die Verbindung zum Gegenüber betrifft. Dazu gehört zum Beispiel der **Connection Monitor**, welcher einen Verbindungsabbruch bemerkt oder auch eine allgemeine “Quit”-Methode, welche sicherstellt, dass der Client korrekt und vollständig abgemeldet wird.

Erwähnenswert ist sicherlich auch die **SendToClient**-Klasse. Von hier aus kann überall eine Instanz kreiert werden, über die dann Befehle an den Client gesendet werden können. Dabei kontrolliert die Klasse selbständig mithilfe des Protocol Enums, dass die gesendeten Befehle korrekt und erlaubt sind. Diese Funktionalität als eigene Klasse anzubieten und nicht direkt auf den Out-Stream im richtigen **ClientHandler**-Objekt zugreifen zu müssen, hat uns einiges erleichtert.

Die Clientseite:



Im Vergleich zum Server ist der Aufbau des Clients einiges einfacher und übersichtlicher. Da es nur einen Client gibt und dieser auch nur eine Verbindung zu nur einem Server hat, fällt schon viel Komplexität weg. Es gibt lediglich zwei Empfängerklassen, einmal die **ClientReceive**-Klasse, welche die Verbindung zum Server hält und Daten empfängt, und einmal die **ConsoleIn**-Klasse, welche Inputs aus der Konsole entgegen nimmt.
Die Daten dieser beiden Klassen werden dann über die **ClientReceive**-Klasse mit dem Protokoll abgeglichen und anschliessend innerhalb des Clients an den richtigen Ort weitergeleitet.

Das Gui ist ähnlich aufgebaut wie der **ServerManager** mit den **ClientHandler**. Es gibt die Hauptklasse **Main**, welche den Überblick über alle **Scenes**, **Stages**, **Pop Up's** und den entsprechenden **Controllern** hat. So kann auch die **ClientReceive**-Klasse ganz einfach über das **Gui-Main** die entsprechenden **Controller** abfragen und anschliessend deren Methoden ansprechen.